Learning Generalisable Bimanual Robotic Manipulation with Adaptation and Vision-Language Model Feedback

Seung-Bin Joo

Wadham College

University of Oxford

Submitted for the degree of

Master of Engineering in Engineering Science

Trinity 2025

Acknowledgements

I am deeply grateful to many individuals and institutions whose support made this thesis possible. First and foremost, I would like to thank my supervisor Dr. João F. Henriques for his continued guidance, ideas, and feedback, which were central to the progress and success of my thesis. I am grateful to the Visual Geometry Group, University of Oxford for welcoming me as a Master's student researcher and for providing the resources that enabled me to carry out my research.

I am extremely grateful to the Kwanjeong Educational Foundation, whose generous support has made my academic journey at the University of Oxford possible. Lastly, I thank my parents, Kyoung Chyan Joo and Bok Keum Choi, for always always being there for me and helping me grow.

Abstract

Generalisable robotic manipulation is a key challenge in robot learning. For robots to be successfully deployed in diverse real-life manipulation scenarios, they must adapt to variations in object geometry, density, friction coefficient, and external disturbance forces on the robot. On top of generalisable manipulation, to interact with humans effectively, robots must not only understand natural language task descriptions, but also visually and semantically understand the scene and objects being manipulated. Lastly, for many tasks in practical settings, we require robots to utilise bimanual (i.e. two-arm) capabilities to achieve complex human-like behaviours.

We use Rapid Motor Adaptation (RMA)—a reinforcement learning (RL) technique—to tackle generalisable bimanual manipulation. With the RMA method, our bimanual manipulator arms learn to adapt in real-time to variations in object configurations by predicting privileged environment information. We then augment bimanual RMA by combining it with Reinforcement Learning from Vision-Language Foundation Model Feedback (RL-VLM-F). This allows us to imbue the bimanual robots with broad semantic understanding by using vision-language models (VLMs) pre-trained on internet-scale data to automatically learn RL reward functions. We call this final pipeline **Bi**manual **G**ene**R**alisable manipulation with **A**daptation and vi**S**ion-language **P**references (Bi-GRASP).

Bimanual RMA and Bi-GRASP were built and tested in ManiSkill 3, an open-source robot simulation and training framework. We show that bimanual RMA can solve challenging two-arm manipulation tasks involving diverse YCB objects, with success rates competitive with or superior to state of the art techniques and baselines. Moreover, we demonstrate that Bi-GRASP can learn manipulation skills involving object generalisability, vision-language understanding and bimanual coordination.

Contents

1 Introduction

1.1 Motivation and Project Objectives

Despite remarkable progress in artificial intelligence (AI) over the past decade [1]–[5], creating general-purpose robots that achieve complex manipulation tasks, semantically understand the surrounding environment, and exhibit broad generalisation over everyday human-achievable tasks remains a challenge. Such general-purpose robots would have wide-ranging impacts on society by enabling assistance in physically demanding and time-consuming tasks such as household automation and industrial manufacturing. The aim of the project is to make advancements towards such general-purpose robots, specifically, by achieving three main objectives:

1. **Generalisable manipulation:** For successful deployment in real-life scenarios, robots must be able to perform manipulation tasks (e.g. grasping an object and placing it at some goal position) in diverse environment configurations. This means they must adapt their manipulation behaviour to general objects with potentially varying geometries, friction coefficients, scales, densities, etc., instead of learning manipulation skills for a single specific configuration. General manipulation also requires adapting to changing levels of noise and disturbance (e.g. to the robot joint positions, object position and object orientation) that a robot might encounter in any general real-life environment.

2. **Broad vision-language understanding**: Natural language is one of the most intuitive, and therefore ideal, methods for a human to interact with and command a robot. We aim to explore algorithms that allow us to specify a task to a robot via a text description. To that end, the robot must not only be able to understand the text description of the task, but also use its perception to visually understand the environment. This entails understanding object semantics (e.g. object identification such as "this is a cup"), spatial semantics (e.g. spatial relations such as "the cup is on the table") and task-relevant understanding (e.g. the meaning of actions such as "pour water into cup"). We aim to achieve *broad* vision-language understanding, enabling the robot to semantically interpret diverse objects and

environments, rather than relying on narrowly curated categories or task-specific knowledge.

3. **Bimanual manipulation:** Compared to single-arm robot settings, the number and complexity of achievable manipulation tasks greatly increases with bimanual robot setups (two-arm robots). Many tasks relevant to humans naturally require two arms (e.g. lifting objects, handing over objects from one hand to another, opening a bottle, etc.). Thus, we aim to achieve objectives 1 and 2 for bimanual robotic manipulation tasks.

In summary, the goal of the project is to develop algorithms and a system that imbues bimanual robots with broad vision-language understanding and the ability to learn generalisable manipulation, as visualised in Figure 1. As an overview, we will achieve this goal using deep reinforcement learning (RL), a promising approach in robot learning.

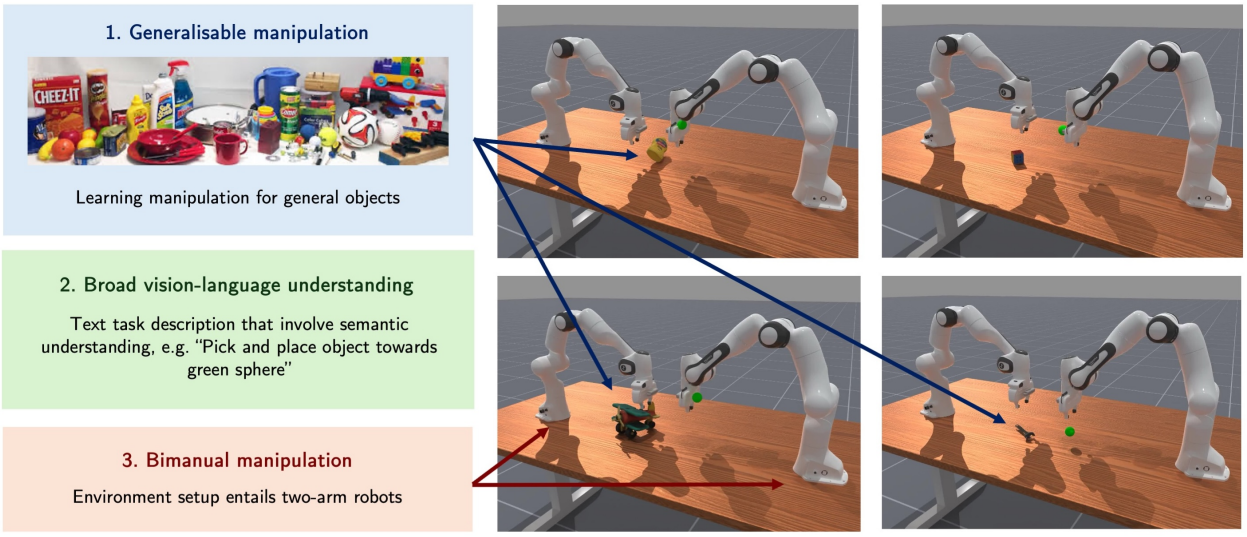

Figure 1: The three project objectives. The image in the blue box shows everyday objects from the YCB object dataset [6]. The 4 objects shown on the right are sampled from the YCB dataset.

1.2 Structure of this Report

The remainder of this report has 5 main sections:

- Section 2 is a literature review of related works that are relevant to the project. This provides essential context and background to the research in this project.
- Section 3 provides the preliminary theory in RL that is the foundational to understanding the technical sections that follow.

- Section 4 explores achieving generalisable bimanual manipulation using Rapid Motor Adaptation (RMA). We evaluate bimanual RMA using the experimental setups described in Section 4.3, and present the corresponding results in Section 4.5.
- Section 5 then augments the bimanual RMA pipeline by adding vision-language model (VLM) feedback. The results for the full pipeline, termed Bi-GRASP, are discussed in Section 5.6.
- Section 6 provides a summary of the contributions, as well as future work.

2 Related Works and Background

2.1 Learning Generalisable Robot Manipulation

In the field of robot learning, two main approaches exist for training robots that are general both in terms of manipulation capability and vision-language understanding: imitation learning (IL) and reinforcement learning (RL).

In IL, the robot policy is learned by mimicking expert demonstrations [7]. IL agents such as *CLIPort* cleverly combine a pre-trained multi-modal CLIP model [8] with TransportNets [9] to learn language-conditioned manipulation tasks. Works such as *PerAct* have utilised transformers as IL agents to encode language goals and RGB-D voxel observations. More recently, a plethora of research, including *RT-1* [10], *RT-2* [11], *RDT-1B* [12], *OpenVLA* [13] and *Octo* [14], has been devoted to fusing large foundation models (e.g. with diffusion transformer-based architectures) with large amounts of robot demonstration data [15] to create multi-billion parameter generalist robot IL policies. While IL for robot manipulation has shown impressive results, there are significant limitations with this approach. Expert robot demonstration data, i.e. recorded examples of a task being performed correctly (often by a human), is expensive to collect. For a truly generalist robot, one would need to collect a prohibitively large amount of expert demonstration data, as IL agents often do not generalize well under distribution shifts (e.g. grasping objects that were not in the expert demonstration data). Moreover, IL agents also passively mimic human expert data without exploring alternative strategies that might differ or potentially be superior to human behaviour.

For those reasons, in this project, we focus on the RL approach, where agents learn manipulation

skills by maximizing cumulative rewards they receive from interacting with an environment. Many works have explored deep RL for manipulation, including model-free approaches for highly dextrous grasping tasks of general objects [16], methods that use Q-attention to improve sample efficiency [17], and even works that tackle the manipulation of deformable objects [18], [19].

Importantly, much of RL research for manipulation can be categorised as Sim-to-Real RL, where a robot policy is trained in a simulation environment (e.g. MuJoCo, Isaac Gym, Drake, ManiSkill), and subsequently transferred to a real robot. This enables safe and fast learning in simulation as real robots are not damaged during training and parallel training environments can be simulated at high speeds. However, the sim-to-real gap, i.e. mismatch between simulation and the real-world, poses a great challenge and is a primary cause for poor real-world performance of RL agents. This sim-to-real gap can be attributed to many factors including: the difference between the physical robot and the robot model used in simulation, the varying environmental conditions in the real-world (e.g. constantly changing values of friction, disturbance forces, contact forces, etc.) and the inaccuracy of the physics simulator relative to real-world physics [20]. A common approach to tackling the sim-to-real transfer is domain randomisation, where the simulation parameters (e.g., textures, lighting, masses, friction) are randomised during training, so the policy learns to be robust to variations it may encounter in the real world [21], [22]. Domain randomisation prevents robot policies from overfitting to a single simulation setup, but they do not explicitly allow policies to adapt online (in real-time) to mismatches between training and test setups. While the transfer to real robots is not a focus of this report, Sim-to-Real RL and domain randomisation are important for Section 4 and 4.1.5 where adaptation to various environment configurations is discussed.

Lastly, we note that many RL manipulation works have yet to explore combining generalisable manipulation with broad semantic understanding. This is one reason why uniting generalisable manipulation with vision-language understanding, specifically through RL, is a focus of this thesis.

2.2 Bimanual Manipulation

While bimanual manipulation is critical to many human real-world tasks (e.g. hanging clothes on a hanger, opening a jar, etc.), it is also more challenging than single-arm manipulation due

to the higher dimensional action space. Bimanual manipulation has a rich history in robotics, with early works tackling two arm robot tasks using optimization techniques [23]. More recently, bimanual manipulation research efforts have shifted towards RL and IL approaches. For instance, *ALOHA Unleashed* succeeded in challenging real-world bimanual tasks, such as tying shoelaces, by using large scale data collection and an IL diffusion policy. Other IL approaches use voxel-based representations and transformer architectures, while RL methods apply Sim-to-Real techniques along with state-of-the-art RL algorithms such as Proximal Policy Optimization to tackle similarly challenging bimanual manipulation tasks [24]–[29]. For some bimanual tasks, it has been shown that parametrising the policy so that one arm takes on a stabilizing role while the other performs the primary task can improve performance [30].

2.3 Reward Learning with Large Pre-trained Models and Feedback

One promising method to imbue RL agents with vision-language understanding is to learn the reward model with large pre-trained models. A reward model is a function that assigns scalar values to an agent's behaviours, guiding the RL process by indicating how desirable each action or outcome is with respect to a given task. This reward model typically needs to be manually specified or hard-coded for each task by defining explicit criteria that the agent must optimize. This process, called reward shaping, requires domain expertise, careful tuning, and trial-and-error [31]. As tasks grow in complexity, become language-specified, and take place in increasingly open-ended and visually rich environments, writing an appropriate reward function becomes hard to scale (each new task needs a new reward design) and brittle (small semantic changes to task description can break the reward function logic). Thus, manually designing reward models becomes intractable at scale, especially in the context of generalist robots that need to perform many semantic tasks in diverse environments. This motivates learning reward models using large pre-trained models (e.g. large language models and vision-language models), where pre-trained models can interpret language-based task descriptions, provide agents with semantic understanding and automate the reward shaping process.

Research in this domain is active, with works demonstrating the use of large language models

(LLMs) to directly write dense reward function code [31]–[33]. However, this approach can struggle with tasks that require more ambiguous and challenging reward functions (e.g. for a cloth folding task, it would be difficult to generate reward code for tracking multiple points on a deformable cloth). For such tasks, performance can be improved by using vision-language models (VLMs) as reward functions, which compute the alignment between the agent's image observations and the textual task descriptions [34], [35]. Experiments show that computing text-image alignment as a reward score can be noisy, so another line of works choose to use preference labels (i.e. binary feedback) instead. This research builds upon Reinforcement Learning from Human Feedback (RLHF) [36] and extends to Reinforcement Learning from AI Feedback (RLAIF) [37], [38], where reward models are learned based on preference labels over an agent's observation-action pairs. This preference-based RL is the method we select to further explore, and is detailed in Section 5.

3 Preliminary Theory

3.1 Reinforcement Learning (RL)

Reinforcement learning (RL) is a machine learning paradigm where agents interact with an environment, and are trained to make sequences of actions by maximising a cumulative reward [39]. In RL, for a given time-step t, agents observe a state s_t, execute an action a_t, receive a scalar reward r_t, and transition to the next state s_{t+1}. The RL objective is to learn a policy $\pi(a|s)$—a mapping from states to action—that maximises the expected return R_t, i.e. the discounted sum of future rewards:

$$R_t = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right], \tag{1}$$

where $\gamma \in [0, 1)$ represents the discount factor. In this report, we will use a stochastic policy, meaning the policy $\pi(a|s)$ is a probability distribution over possible actions, given a state.

3.1.1 Markov Decision Process (MDP)

RL problems, including robot manipulation tasks, are typically mathematically formulated into Markov Decision Processes (MDPs). An MDP is defined by a a 5-tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$:

- $\mathcal{S}$ is the set of possible environment states. In our robot manipulation context, the state s includes the joint positions of the robot manipulator arm and the position and orientation of the objects in the environment, as well as simulation environment parameters such as the friction coefficients of joints or surfaces and the mass of the target objects.

- $\mathcal{A}$ is the set of possible actions that an agent can take. The definition of the action space in our robot manipulation setting depends on the controller mode specified in the simulation environment (see Section 4.3.3 and Section 5.4.2), but for now it suffices to think of actions a as target joint positions in the robot's configuration space.

- $\mathcal{P}(s'|s,a)$ is the transition function defining the probability of transitioning to next state s' given the state s and action a. For our simulated robot manipulation, $\mathcal{P}$ is governed by the SAPIEN physics simulator (see Section 4.3.1).

- $\mathcal{R}(s,a)$ is the reward function which gives the agent an immediate reward r_t for executing an action a at state s. For instance, in a pick and place manipulation task, the reward function may output a large scalar for a robot being near an object and executing a grasping action.

- $\gamma \in [0,1)$ is the discount factor that determines how future rewards are weighted relative to immediate rewards.

In practical settings, the robot perceives the true state s through partial observations o from sensors. However, in simulation, we have access to the full ground-truth state of the environment.

3.1.2 Policy Gradient Methods

In this report, we use a class of RL algorithms called policy gradient methods to solve the robot manipulation MDP. In deep RL policy gradient methods, the policy π_θ is typically parametrised using a multilayer perceptron (MLP), i.e. a neural network. The approach is to then directly optimise the policy by updating the policy network parameters θ based on gradient ascent, where the optimisation objective $J(\theta)$ is the expected return:

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta) \tag{2}$$

To optimise (via gradient ascent) the expected return objective $J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T}\gamma^t r(s_t, a_t)\right]$, where $\tau = (s_0, a_0, ..., s_T)$ is a trajectory sampled using policy π_θ, we require the gradient of the objective $J(\theta)$. This is given by the Policy Gradient Theorem [39]:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a_t|s_t) \cdot R_t\right] \tag{3}$$

Thus, we can sample batches of trajectories τ, and use the expectation expression above to compute an estimate of the gradient of the objective $\nabla_\theta J(\theta)$. Note that in modern policy gradient methods, including those used in this report (see Section 3.2.1 and 3.2.2), instead of using the return R_t for gradient estimation, *actor-critic* methods and its variants learn not only an actor (policy), but also a critic network that estimates the advantage function $A(s_t, a_t)$, i.e. the relative benefit of taking a particular action in a given state compared to the expected value of that state. Using the advantage function $A(s_t, a_t)$ instead of relying on the potentially noisy raw return R_t reduces the variance of the gradient estimate and stabilises training, as $A(s_t, a_t)$ reflects how much better or worse an action is compared to the expected value of the state.

3.2 On-policy and Off-policy Methods

RL algorithms where the policy updates are based only on trajectories sampled using the current policy are referred to as on-policy. Since the policy collecting experiences and the policy being optimised are the same, on-policy algorithms can be more stable, but also less sample-efficient, since new trajectory data must be collected for every policy update.

In contrast, in off-policy methods, the policy gathering trajectory data (the "behaviour policy") can be different from the policy that is being updated (the "target policy"). The behaviour policy may be more exploratory and also store its trajectories in a large *replay buffer*. The target policy can sample from this buffer and reuse trajectories for its updates, achieving greater sample efficiency. However, the mismatch between the policy being updated and the data collecting policy can lead to more complex stability issues during training.

Finally, we introduce two main RL algorithms that are integral to fully understanding the

broader RL pipelines presented in this report. A brief overview of their key concepts is offered, though the detailed inner workings of these algorithms are not the primary focus of this report.

3.2.1 Proximal Policy Optimisation (PPO)

Proximal Policy Optimisation (PPO) is an on-policy policy gradient algorithm designed to improve training stability [40]. PPO limits the size of each policy update—ensuring that we take the largest possible policy improvement step during the gradient ascent without deviating so drastically that it leads to instability and a collapse in performance. The key idea to achieve this is to use a clipped objective $L^{\text{CLIP}}(\theta)$:

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_t \left[\min \left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} \hat{A}_t,\ \text{clip} \left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}, 1-\epsilon, 1+\epsilon \right) \hat{A}_t \right) \right] \tag{4}$$

where $\hat{A}_t$ is the estimated advantage at time step t and ϵ is the clipping range hyperparameter (e.g. 0.1) that defines how far away the new policy is allowed to be relative to the old policy. For instance, if the advantage $\hat{A}_t$ for a certain state-action pair (s_t, a_t) is positive, the policy may update from $\pi_{\theta_{\text{old}}}(a|s_t)$ to $\pi_\theta(a_t|s_t)$, making action a_t more likely and hence increasing the objective. However, if the update (measured by the probability ratio $\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$) is too large, i.e. larger than the positively clipped ratio $\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}(1+\epsilon)$, then the min term limits how much the objective can increase. When the advantage is negative, the same logic applies to the negatively clipped ratio $\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}(1-\epsilon)$. PPO utilises an actor-critic architecture, where the actor maximises the clipped objective L^{CLIP} and the critic is trained via mean squared error (MSE) regression to bootstrapped value targets, typically computed using Generalized Advantage Estimation (GAE) [40].

3.2.2 Soft Actor Critic (SAC)

Soft Actor Critic (SAC) is an off-policy policy gradient algorithm whose key feature is entropy regularization [41]. In SAC, the agent is trained to maximize the expected return and the policy entropy, a measure of uncertainty in the policy's action distribution. The SAC objective $J(\pi)$ is:

$$J(\pi) = \sum_t \mathbb{E}_{(s_t,a_t)\sim\mathcal{D}} \left[r(s_t, a_t) + \alpha \mathcal{H}(\pi(\cdot|s_t)) \right] \tag{5}$$

The entropy term $\mathcal{H}(\pi(\cdot|s_t)) = -\mathbb{E}_{a_t\sim\pi}[\log\pi(a_t|s_t)]$ aids the policy in exploration and aims to prevent early convergence to a poor local optimum. The temperature coefficient α determines the trade-off between reward and entropy maximisation. In many cases, including our robot manipulation case, SAC is implemented with a large replay buffer $\mathcal{D}$ that allows the agent to sample and reuse experiences (s_t, a_t) in an off-policy manner. Lastly, SAC is an actor-critic method, so the actor learns to maximise the SAC objective, while the critic learns to predict the Q-function (i.e. the expected cumulative reward given a state-action pair). Since Q-functions can be overly optimistic in SAC, two critic networks are learned and the minimum of the two is used in the critic update. The critic networks are updated towards a target y based on the Bellman backup—the idea that: (value of state-action pair) = (immediate reward) + $\gamma \times$ (value of the next state)—with an additional term $\alpha\log\pi(a_{t+1}|s_{t+1})$ to account for entropy regularization [41]:

$$y = r(s_t, a_t) + \gamma\mathbb{E}_{a_{t+1}\sim\pi}\left[\min_{i=1,2} Q_{\theta_i}(s_{t+1}, a_{t+1}) - \alpha\log\pi(a_{t+1}|s_{t+1})\right] \tag{6}$$

4 Reinforcement Learning with Adaptation

4.1 Rapid Motor Adaptation (RMA) Overview

In order to tackle generalisable bimanual manipulation, we consider Rapid Motor Adaptation (RMA) [20], [42], an RL approach that allows robots to adapt online (real-time) to diverse environment configurations, including variations in object geometry, density, friction coefficient, etc.

Figure 2 demonstrates that RMA is divided into two phases of training: base policy training and adapter training. During base policy training, a policy π is trained in simulation while having access to privileged environment parameters e and state information s_t. The vectors e and s_t include information, such as object size, density, friction, as well as, object orientation, which are available via the simulation. Using an environment encoder $\mu(e, s_t)$, the environment information is encoded into a latent feature vector z_t which we call the environment embedding. The base policy is then conditioned on this environment embedding, along with other goal features g and observations o it typically requires. We train the base policy with high amounts of domain randomization (i.e.

randomly and heavily vary object identities and the environment parameters), but since the policy has privileged knowledge of the environment configuration, it learns behaviour such that it can adapt to diverse target objects and manipulation settings.

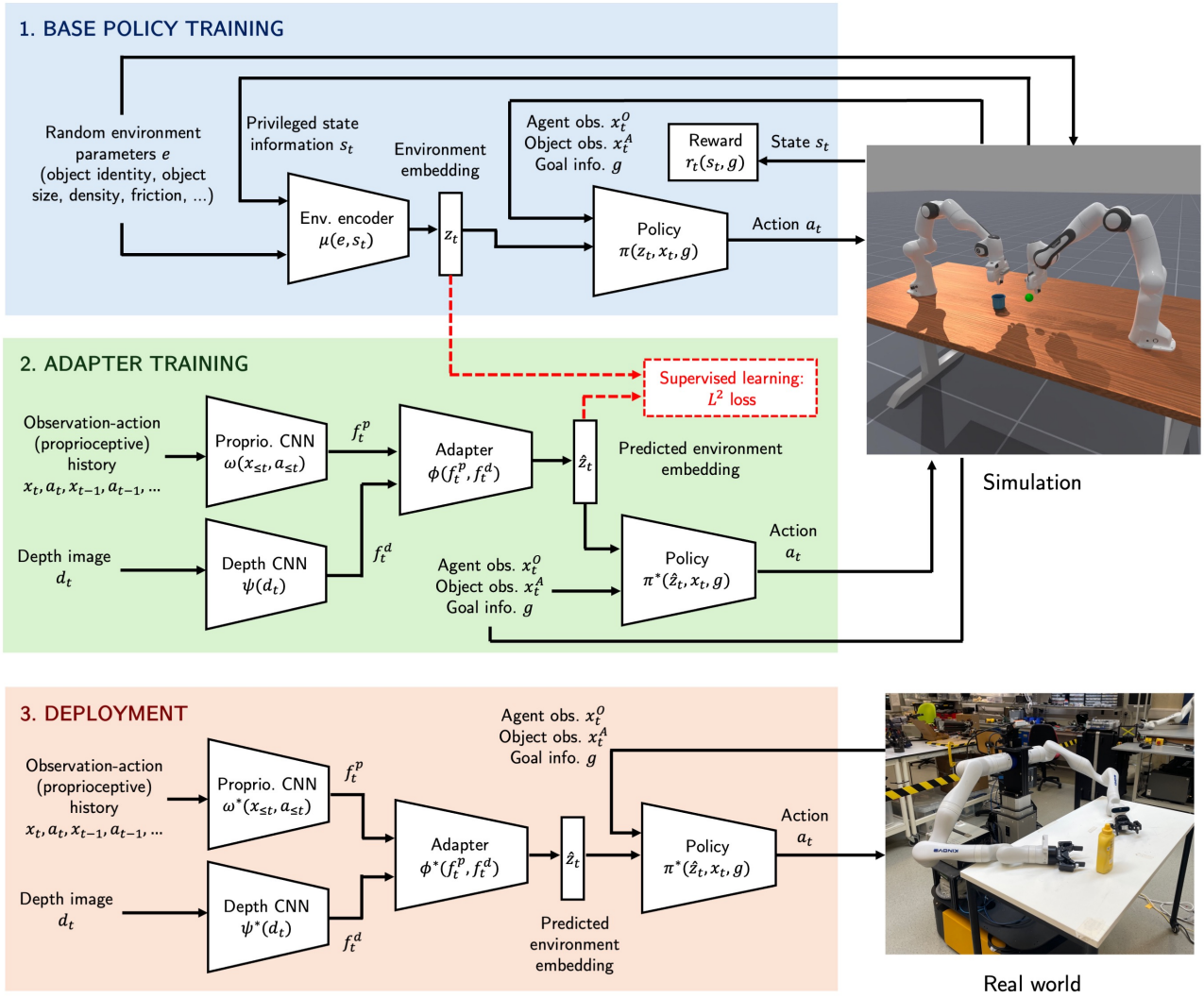

Figure 2: Overview of the Rapid Motor Adaptation (RMA) pipeline. During the first base policy training phase, the environment encoder $\mu(e, s_t)$ embeds the privileged information from simulation into the extrinsic vector z_t. The policy $\pi(z_t, x_t, g)$ outputs an action, conditioned on the environment embedding z_t, observations o_t (e.g. robot joint angles) and goal information g (e.g. desired goal pose for an object being grasped). The environment encoder $\mu(e, s_t)$ and policy $\pi(z_t, x_t, g)$ are jointly trained under high domain randomization using RL to maximise a cumulative reward (Equation 1), which is defined by a task (e.g. "hand over an object and place it at goal position"). In a second phase, the trained base policy $\pi^*(z_t, x_t, g)$ is frozen and the adapter $\phi(f_t^p, f_t^d)$, proprioception CNN $\omega(x_{\leq t}, a_{\leq t})$ and depth CNN $\psi(d_t)$ are trained jointly to predict the environment embedding $\hat{z}_t$ by MSE regression to the ground truth environment embedding z_t. The base policy during the adapter training stage uses the predicted environment embedding $\hat{z}_t$. In the final deployment phase, we evaluate actions using the trained adapters and base policy.

However, the base policy cannot be simply deployed, because in most practical settings, we do not have access to privileged environment information (this was only available via the simulator). Thus, RMA utilises an *adapter* which learns to estimate the environment embedding z_t with readily-available inputs, such as the robot's proprioceptive history (i.e. history of action and joint position pairs). As shown in Figure 2, a proprioceptive convolutional neural network (CNN) ω, depth image CNN ϕ and adapter ϕ generate a prediction for the environment embedding $\hat{z}_t$.

During deployment (see Figure 2), we use the learned adapter ϕ^* and optimised base policy π^* conditioned on the predicted environment embedding $\hat{z}_t$ to perform highly adaptive manipulation for different objects, environments and scenarios. RMA was first applied to quadruped robots [20] but has also since been shown to work with bipedal [43] and arm robots [42].

4.1.1 Base Policy

The base policy $\pi(x_t, z_t, g)$ and environment encoder $\mu(e, s_t)$ are jointly trained by maximising the RL objective (Equation 1). We formulate this optimisation into the following equation [42]:

$$\pi^*, \mu^* = \underset{\pi,\mu}{\operatorname{argmax}} \, \mathbb{E}_{\substack{e\sim\mathcal{E} \\ g\sim\mathcal{G}}} \left[\mathbb{E}_{s_{t+1}\sim P_e(\cdot|s_t,a_t)} \left[\sum_{t'=0}^{T-1} \gamma^{t'} r(s_{t'}, g) \right] \right] \tag{7}$$

$$\text{with } s_t \sim P_e(s_t, a_t), \quad x_t = o(s_t), \quad a_t = \pi(x_t, z_t, g), \quad z_t = \mu(e, s_t)$$

The expression looks to maximise the expectation of the RL objective $\sum_{t'=0}^{T-1} \gamma^{t'} r(s_{t'}, g)$, where T is the maximum simulation length. The expectations are taken with respect to $\mathcal{E}$, the distribution of environment parameters e (determined by domain randomization of physical parameters such as object geometry, friction, density, etc.), and $\mathcal{G}$, the distribution of manipulation goals g (e.g. for a pick and place object task, this would be desired goal pose of the object). The transition probability $s_t \sim P_e(s_t, a_t)$ governs how the current state s_t and action a_t map to the next state, and is governed by the ManiSkill and SAPIEN robot simulation framework detailed in Section 4.3.1 and 4.3.3. The simulator, and hence P_e, is parametrised by the environment parameters e. The observation model $x_t = o(s_t)$, also handled by the simulator, represents the robot not having full access to the true state, and thus receiving observations x_t, such as the position of the object. A detailed description of the components of the state, observations, environment parameters and

privileged state information is provided in Section 4.1.2.

As mentioned, the environment encoder outputs an environment embedding via $z_t = \mu(e, s_t)$ and the actions are then given by the environment embedding-conditioned policy $a_t = \pi(x_t, z_t, g)$. The purpose of the environment encoder is to provide a compressed and learned latent feature vector that encompasses environment information that is critical for the robot in adapting and succeeding in the manipulation task. The environment encoder and base policy are MLPs (architectures described in Section 4.3.5). Finally, we use PPO (described earlier in Section 3.2.1) to optimise Equation 7 in an on-policy manner.

Base policy training is outlined in Algorithm 1. Training takes place in simulation with N_{env} parallel environments. For each environment, domain randomisation is initiated when $\text{step}_{\text{global}}$ exceeds $\text{step}_{\text{rand\_start}}$. To gradually expand the randomisation range from default environment parameters e_{default} to a target interval $[e_{\min}, e_{\max}]$, we interpolate the bounds using a linear ramp factor $\beta \in [0, 1]$ based on the global step. This ensures that early training uses a fixed set of environment parameters, while later training samples uniformly from an increasingly wide range. Each episode begins with an environment reset, where the scene is reloaded and the object is randomised. Then, for T time-steps, we step through with the environment encoder and base policy, storing transitions (x_t, a_t, r_t, x_{t+1}) into a buffer D_{PPO}. When all environment episodes finish we backpropagate and update the base policy and environment encoder jointly using PPO.

4.1.2 States, Observations and Environment Parameters

In this section, we clarify the distinction between states, privileged state information, observations and environment parameters for our robot manipulation context.

- **State** s is all physical variables describing the simulation environment at a time step.
- **Privileged state information** s_t denotes the physical variables that we encourage the model to encode as they may be useful for the policy when achieving the manipulation goal. In our robot manipulation setting, we include the 4D quaternion describing the object's orientation and the contact impulse measured between the robot's grippers and the object.
- **Observations** x_t are measurable variables that a robot has available in simulation but also

could obtain during deployment. For example, in our context, we treat the angular positions and velocities of the robot's joints and gripper as observations (during deployment, this could be measured through rotary encoders). We also include object 3D position as an observation, as this can be obtained through a standard vision-based object pose estimator [44].

- **Environment parameters** e are the parameters used to instantiate the simulation environment and are constant throughout an episode. In our simulation, these are the object geometry, scale, friction and density, which serve as inputs to the environment encoder.

Algorithm 1 Base Policy Training (On-Policy)

Initialise weights of base policy π and environment encoder μ;
Initialise empty replay buffer D_{PPO};
Initialise default environment parameters e_{default} and their randomisation ranges $e_{\text{min}}, e_{\text{max}}$
for $0 \leq \text{itr} \leq N_{\text{itr}}^{\text{PPO}}$ **do**
  for $0 \leq i < N_{\text{env}}$ **do**
    if $\text{step}_{\text{global}} \geq \text{step}_{\text{rand\_start}}$ **then**
      $\beta \leftarrow \min\left(\frac{\text{step}_{\text{global}} - \text{step}_{\text{rand\_start}}}{\text{step}_{\text{rand\_end}} - \text{step}_{\text{rand\_start}}}, 1\right)$
      $\text{low} \leftarrow e_{\text{default}} - \beta\left(e_{\text{default}} - e_{\text{min}}\right)$
      $\text{high} \leftarrow e_{\text{default}} + \beta\left(e_{\text{max}} - e_{\text{default}}\right)$
      $e \sim \mathcal{U}(\text{low}, \text{high})$
    else
      $e \leftarrow e_{\text{default}}$
    end if
    $x_0 \leftarrow \text{envs}[i].\text{reset}()$;
    for $0 \leq t \leq T$ **do**
      $z_t \leftarrow \mu(e, s_t)$;
      $a_t \leftarrow \pi(x_t, z_t, g)$;
      $x_{t+1}, r_t \leftarrow \text{envs}[i].\text{step}(a_t)$;
      Store (x_t, a_t, r_t, x_{t+1}) in D_{PPO};
      $\text{step}_{\text{global}} \leftarrow \text{step}_{\text{global}} + 1$
    end for
  end for
  Use PPO to update π and μ (see Equation 5 and 7);
  Empty D_{PPO};
end for

4.1.3 Object and Category Embeddings as a Proxy for Object Geometry Knowledge

To successfully grasp an object, robot manipulators must infer its geometry. We do not explicitly pass an object geometry description into the environment encoder. Instead, we implicitly

encode object geometry knowledge using learnable embedding networks. Given a list of different objects IDs (e.g. 'cup_01', "cup_02", ..., "ball_01", "ball_02", ... "object_i"), any of which may be randomly sampled for the manipulation task, we convert these objects into indices (0, 1, 2, ...) and pass the indices through an embedding neural network that learns e_{obj} a dense continuous vector representation of each object type [42]. Similarly, we also learn an embedding e_{category} for the object category IDs ("cup", "ball", ..., "category_j"). The object embedding e_{obj}, category embedding e_{category}, and remaining physical environment parameters e_{physical} (i.e. scale, friction coefficient and density) are concatenated together and form the inputs to the encoder:

$$e = \text{concatenate}\,(\, e_{\text{obj}},\, e_{\text{category}},\, e_{\text{physical}}\,) \tag{8}$$

4.1.4 Adapter

With base policy training complete, we now have a policy that has learned to utilise privileged environment information to perform general object manipulation. However, since this privileged information is accessible only through simulation, in practice, we must learn an adapter that predicts the environment embedding z_t. RMA approaches this by using the proprioception history, i.e. an observation-action history $(x_{\leq t}, a_{\leq t}) = x_t, a_t, x_{t-1}, a_{t-1}, ..., x_0, a_0$, to predict the privileged environment information such as object density, friction, etc. This is feasible because discrepancies between intended actions and the resulting motion provide implicit cues about underlying physical properties of the environment. Intuitively, this is analogous to a real-world scenario in which a person attempts to lift a box under the incorrect assumption that it is heavy. As a result, they apply a large upward force, causing the box to accelerate rapidly. Through proprioceptive feedback—i.e. the sequence of joint torques, positions, and applied forces over time—the individual infers that the box is lighter than expected. Hence, the agent implicitly estimates privileged information such as object mass based on the history of its observations and actions. As shown in Figure 2, $(x_{\leq t}, a_{\leq t})$ is processed by a 1D proprioception CNN ω to capture temporal correlations and the output is passed to the adapter network ϕ that subsequently predicts the environment embedding $\hat{z}_t$.

Importantly, for our manipulation task, it is difficult for object geometry information (which is part of the information we encode into the environment embedding z_t) to be successfully predicted

from proprioceptive history alone, especially in the time steps prior to the robot arm contacting the object. This necessitates a visual component that can aid in inferring object identity and geometry information. Therefore, low-resolution depth cameras mounted on the wrists of the robot arms, are used to better predict physical geometry-related privileged information. As indicated in Figure 2, the depth images are also passed through a CNN ψ, and the output f_t^d is concatenated with the output of the proprioception CNN f_t^p before entering the adapter neural network ϕ.

As illustrated in Figure 2, we jointly train the proprioception CNN $f_t^p = \omega(x_{\leq t}, a_{\leq t})$, depth CNN $f_t^d = \psi(d_t)$ and adapter $\hat{z}_t = \phi(f_t^p, f_t^d)$ by treating it as a supervised learning problem. For a batch of simulated robot trajectories of length T, we minimise the L^2 loss between the predicted environment embedding $\hat{z}_t$ and the ground truth environment embedding z_t computed using the optimised environment encoder $\mu^*(e, s_t)$. Thus, the objective for adaptation learning is summarised by the following expression [42]:

$$\phi^*, \psi^*, \omega^* = \underset{\pi,\psi,\omega}{\operatorname{argmin}} \, \mathbb{E}_{\substack{e\sim\mathcal{E} \\ g\sim\mathcal{G}}} \left[\mathbb{E}_{s_{t+1}\sim P_e(\cdot|s_t,a_t)} \left[\sum_{t'=0}^{T-1} \|\mu^*(e, s_{t'}) - \hat{z}_{t'}\|^2 \right] \right] \tag{9}$$

$$\text{with} \;\; s_0 \sim P_e(s_0), \;\; a_t = \pi^*(\hat{x}_t, \hat{z}_t, g), \;\; x_t = o(s_t), \;\; f_t^d = \psi(d_t), \;\; f_t^p = \omega(x_{\leq t}, a_{\leq t}), \;\; \hat{z}_t = \phi(f_t^p, f_t^d)$$

where all variables retain their definitions from earlier.

We note that in Equation 9, the executed robot actions are generated by the policy conditioned on the predicted environment embedding: $a_t = \pi^*(\hat{x}_t, \hat{z}_t, g)$. Alternatively, it would be possible to instead use the base policy conditioned on the ground truth environment embedding z_t to collect the observation-action history and update the adapter. However, the robot trajectories and observation-action histories gathered would only represent ideal trajectories in which the robot arms optimally predict and adapt to diverse object scenarios.

Instead, we follow the Algorithm 2. The initial setup is similar to base policy training in terms of domain randomisation schedule. Then, for each parallel environment, at each step, we generate a predicted environment embedding $\hat{z}_t$ using the proprioception, depth and adapter networks. With no gradient flowing, the frozen environment encoder produces the ground truth z_t. Crucially, we sample actions a_t from the base policy conditioned on the predicted $\hat{z}_t$ [20]. This allows the adapter

to explore and learn from trajectories that have imperfect adaptation, enabling it to be more robust during deployment to deviations from perfect environment embedding predictions. The network parameters are updated in an *on-policy* manner by computing the mean squared error over all environments and trajectory time steps ($\frac{1}{TN_{\text{env}}} \sum_{(\hat{z}_t, z_t) \in D_{\text{adapt}}} \|\hat{z}_t - z_t\|^2$) and performing gradient descent with respect to the network parameters $\theta_{\phi,\omega,\psi}$.

Algorithm 2 Adapter Training (On-Policy)

Randomly initialise the adapter ϕ, proprio. CNN ω, depth CNN ψ parametrised by $\theta_{\phi,\omega,\psi}$;
Initialise default environment parameters e_{default} and their randomisation ranges $e_{\text{min}}, e_{\text{max}}$
Initialise empty buffer D_{adapt};
for $0 \leq \text{itr} \leq N_{\text{itr}}^{\text{adapt}}$ **do**
  for $0 \leq i < N_{\text{env}}$ **do**
    if $\text{step}_{\text{global}} \geq \text{step}_{\text{rand\_start}}$ **then**
      $\beta \leftarrow \min\left(\frac{\text{step}_{\text{global}} - \text{step}_{\text{rand\_start}}}{\text{step}_{\text{rand\_end}} - \text{step}_{\text{rand\_start}}}, 1\right)$
      $\text{low} \leftarrow e_{\text{default}} - \beta\left(e_{\text{default}} - e_{\text{min}}\right)$
      $\text{high} \leftarrow e_{\text{default}} + \beta\left(e_{\text{max}} - e_{\text{default}}\right)$
      $e \sim \mathcal{U}(\text{low, high})$
    else
      $e \leftarrow e_{\text{default}}$
    end if
    $x_0 \leftarrow \text{envs}[i].\text{reset}()$;
    for $0 \leq t \leq T$ **do**
      $f_t^d \leftarrow \psi(d_t)$;
      $f_t^p \leftarrow \omega(x_{\leq t}, a_{\leq t})$;
      $\hat{z}_t \leftarrow \phi(f_t^d, f_t^p)$;
      with no gradient computation:
        $z_t \leftarrow \mu^*(e, s_t)$;
        $a_t \leftarrow \pi^*(x_t, \hat{z}_t, g)$;
      $x_{t+1} \leftarrow \text{envs}[i].\text{step}(a_t)$;
      Store $(\hat{z}_t, z_t)$ in D_{adapt};
    end for
  end for
  $\theta_{\phi,\omega,\psi} \leftarrow \theta_{\phi,\omega,\psi} - \lambda \nabla_{\theta_{\phi,\omega,\psi}} \frac{1}{TN_{\text{env}}} \sum_{(\hat{z}_t, z_t) \in D_{\text{adapt}}} \|\hat{z}_t - z_t\|^2$;
  Empty D_{adapt};
end for

4.1.5 Deployment

Figure 2 demonstrates that during deployment and evaluation time, we utilise the trained base policy π^*, adapter ϕ^*, proprioception CNN ω^* and depth CNN ψ^*. In this report, our

evaluation experiments take place in simulation (due to time constraints); however, RMA without any modification has been shown to be successful with sim-to-real transfer [20]. In some real-life robot settings, environment embedding $\hat{z}_t$ prediction may be imperfect due to the sim-to-real domain gap. The base policy—which was trained while being conditioned on the perfect environment embedding $\hat{z}_t$—using a poor prediction of $\hat{z}_t$ instead, can cause a drop in performance. In these cases, we note that the base policy π can be fine-tuned using PPO in a third phase of training [43], where the adapter, proprioception CNN, and depth CNN are frozen.

While less crucial in simulation, we note that for real-life deployment, it may be necessary to asynchronously run the adapter models and the base policy. The adapter models are slow as they need to process a proprioception history (of 50 time-steps in our case), as well as a depth image. Therefore, the adapter models can run at a lower frequency (e.g. 10 Hz), while the base policy runs at a higher frequency (100 Hz) simply by conditioning on the most recent $\hat{z}_t$ and using the current observation x_t. This asynchronous execution is feasible because the predicted environment embedding $\hat{z}_t$ does not rapidly and consistently vary during an episode as it represents relatively constant physical environment parameters such as object density, friction, etc.

4.2 RMA with Bimanual Manipulation

As discussed in Section 1.1, we aim to achieve *bimanual* generalisable manipulation. There are many strategies: some that treat the robot arms as two independent agents that can coordinate only through visual perception [29], and others that use a leader-follower paradigm where the action from one agent is passed to the other agent to use for its action prediction [25]. We choose the simple approach of treating the two robot arms as one agent, concatenating the two robot arms' actions, observations (e.g. joint anglular positions and velocities) and privileged state information (e.g. gripper-object contact impulses) into single vectors.

We also mount a depth camera on the wrist of each of the two robot arms. This configuration yields two 32×32 depth images, denoted as d_t^1 and d_t^2, which serve as input to the adaptation models. While alternative approaches such as concatenating these images into a single 64×32 input for a correspondingly scaled depth CNN ω, or employing two independent depth CNNs, were

considered, they were deemed less efficient due to increased memory consumption during training and higher compute demands during deployment. Instead, we opted for concatenating the depth images along the batch dimension and utilising a single square depth CNN. This approach was empirically found to effectively predict privileged environment information from both d_t^1 and d_t^2, while maintaining a more compact architecture.

4.3 Experiment and Setup

We now detail the simulation environment, tasks, and experimental setup used to evaluate our generalisable bimanual manipulation pipeline.

4.3.1 ManiSkill Robot Simulation Framework

For training and evaluating generalisable bimanual manipulation, we use ManiSkill 3, a state-of-the-art open-source robot simulation framework [45]. ManiSkill is built on top of the SAPIEN physics simulator [46] and supports GPU parallelised simulation, significantly expediting training (up to 10 to 1000 times faster at simulation and rendering than other robot simulation platforms with less GPU memory usage) [45]. Importantly, ManiSkill 3 supports hetergenous simulation, meaning parallel environments during RL training can have different scenes and randomisations. This makes it an ideal platform for training agents to manipulate general and diverse objects, where each parallel environment can have a random object.

4.3.2 Custom ManiSkill Tasks

We used ManiSkill 3 task building API to construct 3 custom bimanual manipulation tasks that focus on two-arm coordination and object generalisability:

1. **Two Robot Pick YCB**: Referring to Figure 3, the goal is to pick up an object randomly sampled from the YCB dataset [6] and place it within a threshold around the goal position (indicated by the green sphere). The YCB dataset contains 3D models of daily life objects (e.g. cups, fruits, tools, etc.) that are commonly used as a benchmark in generalisable manipulation research (see Figure 1). The YCB object is outside the right arm's reach,

and the goal position is outside the left arm's reach, thus the two arms must coordinate a handover to place the object at the goal. We randomise the object z-rotation, xy-position and the goal's xyz-position (in addition to the environment parameter randomisation detailed in Section 4.3.4). For training, we modify the dense reward function from ManiSkill's Two Robot Pick Cube task so that it support YCB objects.

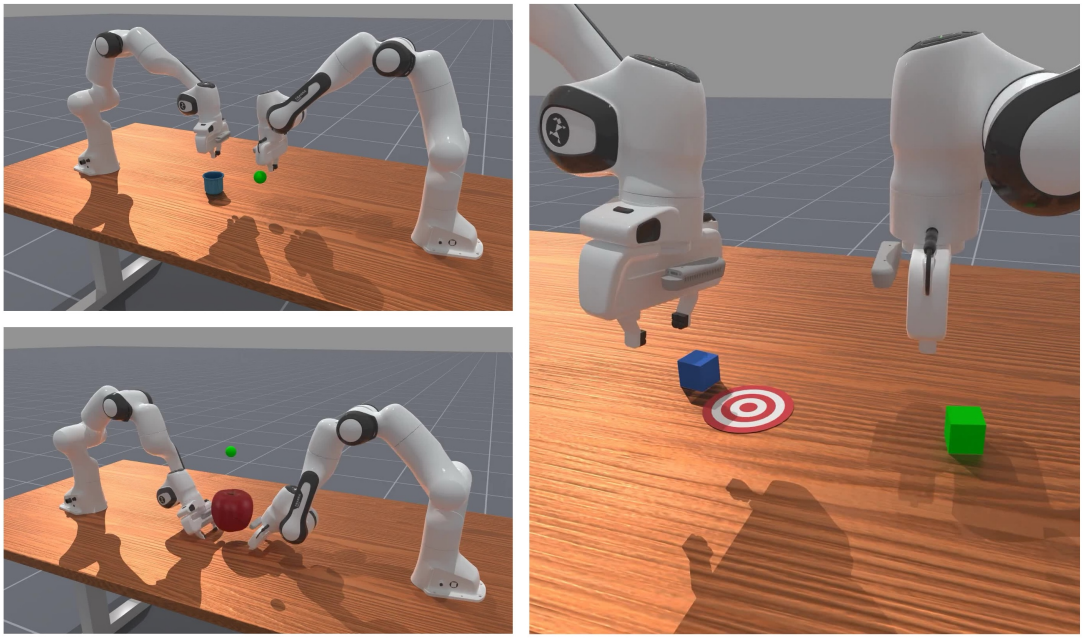

Figure 3: Visualisations of the Two Robot Pick Single YCB (top left), Two Robot Lift YCB (bottom left), and Two Robot Stack Cube (right) tasks.

2. **Two Robot Lift YCB**: Referring to Figure 3, the goal is for the two arms to collaborate in lifting a round YCB object above a target z-position indicated by the green sphere. The YCB object is sampled and scaled such that it is too large for a single robot gripper to grasp, and instead, two robot arms are required to balance and lift the object against gravity. We randomise the z-rotation of the object (as well as environment parameters in Section 4.3.4). In addition, we design a custom dense reward function, shown in Algorithm 3, which provides a smooth reward signal through a staged structure: rewards are given sequentially only if the previous stage is satisfied—grippers under and in contact with the object, minimised object-to-goal distance, and statically holding the object above goal height.

3. **Two Robot Stack Cube**: Referring to Figure 3, the goal is for the right agent to pick and

place the green cube at the target, while the left agent picks and stacks the blue cube above the green cube. Once again, two robot coordination is enforced as the green and blue cubes are non-reachable by the left and right agents, respectively. The z-rotations and xy-positions of the cubes are randomised, and the target is randomly placed at a reachable location along the midline between the two agents. The success condition is satisfied when the blue cube is above the green cube, the green cube is on the target, and both cubes are not in contact with any gripper. This is a default task from ManiSkill [45], but we introduce additional environment parameter randomisations (see Section 4.3.4).

Algorithm 3 Dense Reward Function for Two Robot Lift YCB task

Get object position $p_{\text{obj}} \in \mathbb{R}^{N\times 3}$, left TCP position p_{left}, right TCP position p_{right};
Compute L2 norm of contact impulses:
$i_{\text{left}} \leftarrow \|\text{impulse}(\text{left\_finger}, \text{cube})\|_2$,
$i_{\text{right}} \leftarrow \|\text{impulse}(\text{right\_finger}, \text{cube})\|_2$;
Detect contact: $b_{\text{left}} \leftarrow (i_{\text{left}} > \epsilon)$, $b_{\text{right}} \leftarrow (i_{\text{right}} > \epsilon)$;
Check if fingers are underneath object: $u_{\text{left}} \leftarrow (p^{(z)}_{\text{left}} < p^{(z)}_{\text{obj}} - 0.03)$, $u_{\text{right}} \leftarrow (p^{(z)}_{\text{right}} < p^{(z)}_{\text{obj}} - 0.03)$;
Define under-contact condition: $b_{\text{uc}} \leftarrow b_{\text{left}} \wedge b_{\text{right}} \wedge u_{\text{left}} \wedge u_{\text{right}}$;
Assign contact reward: $r_{\text{contact}} \leftarrow 2 \cdot \mathbf{1}[b_{\text{uc}}]$;
Compute object-to-goal distance: $d_{\text{goal}} \leftarrow \|p_{\text{goal}} - p_{\text{obj}}\|$;
Compute lift reward: $r_{\text{lift}} \leftarrow 3 \cdot (1 - \tanh(5 \cdot d_{\text{goal}}))$;
Detect if object is above goal height: $b_{\text{above}} \leftarrow (p^{(z)}_{\text{obj}} > z_{\text{goal}}) \wedge b_{\text{left}} \wedge b_{\text{right}}$;
Compute joint velocity: $v_{\text{left}} \leftarrow \|\dot{q}_{\text{left}}[1\text{:}-2]\|$, $v_{\text{right}} \leftarrow \|\dot{q}_{\text{right}}[1\text{:}-2]\|$;
Compute static reward:
$s_{\text{left}} \leftarrow 1 - \tanh(5 \cdot v_{\text{left}})$,
$s_{\text{right}} \leftarrow 1 - \tanh(5 \cdot v_{\text{right}})$,
$r_{\text{static}} \leftarrow \frac{1}{2}(s_{\text{left}} + s_{\text{right}})$;
Initialize total reward: $r \leftarrow r_{\text{contact}}$;
Add lift reward where under-contact is satisfied: $r[b_{\text{uc}}] \leftarrow r[b_{\text{uc}}] + r_{\text{lift}}[b_{\text{uc}}]$;
Add static reward where object is lifted: $r[b_{\text{above}}] \leftarrow r[b_{\text{above}}] + r_{\text{static}}[b_{\text{above}}]$;
if success **then**
$r[\text{success}] \leftarrow r[\text{success}] + 5 + r_{\text{static}}[\text{success}]$
end if
return r

4.3.3 Simulation Setup Details

As shown above in Figure 3, for every task, we use two Franka Emika Panda robot arms, a 7 degree of freedom (DOF) manipulator with high dexterity, equipped with torque sensors at each

joint and a two-finger gripper at the end effector of each arm. We utilise a proportional-derivative (PD) controller to drive the robot motors towards target joint positions. Specifically, we use delta control with respect to the joint positions, meaning that an agent's actions a_t represent the desired *change* relative to the current joint positions.

As visualised in Figure 4, we train the bimanual agents in parallel heterogenous simulation environments. While the bimanual agents are trained concurrently across 256 diverse parallel environments, their performance is evaluated periodically using a separate and larger suite of 512 environments to obtain a more reliable measure of learning. The maximum episode length for the bimanual task is set to 50 steps for the Pick and Lift tasks, whereas it is 100 for the Stack task.

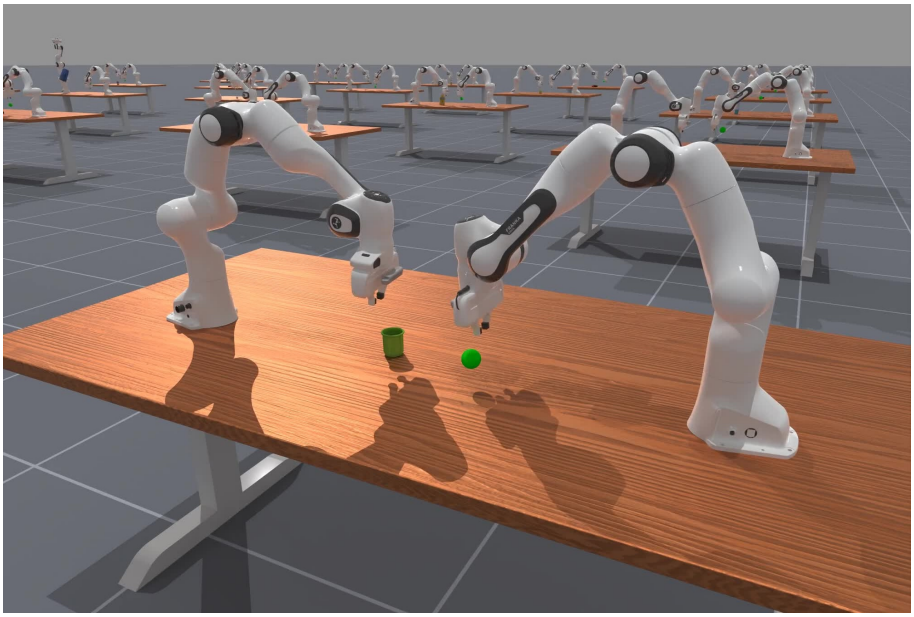

Figure 4: A visualisation of the parallel bimanual manipulation environments in ManiSkill. Note that this ray tracing-enabled render of the parallel environments is for visual demonstration purposes and does not reflect how the environments are simulated during training.

4.3.4 Observation Space and Environment Setup Details

In the bimanual task environments involving YCB objects, the observation x_t space is 56 dimensional. This consists of the joint and gripper positions for the left and right 7 DOF robot arms (18D), joint and gripper velocities (18D), tool center point (TCP) poses of left and right agent (14D) and left and right agent TCP positions relative to the object position (6D). The goal information g passed to the policy is 12 dimensional, composed of the raw goal position

(3D), the goal positions in left and right TCP frames of reference (6D) and the goal position in the object frame of reference (3D). Lastly, the environment parameters e and privileged state information s_t are 75 dimensional in total, split into the object embedding e_{obj} (32D), the object category embedding e_{category} (32D), the object orientation (4D quaternion), object scale (1D), object density (1D), object friction (1D), left and right gripper contact impulses (4D). The Two Robot Stack Cube task does not use object embeddings, but otherwise has similar observations as above but with respect to two cubes (instead of just one object).

In addition to basic task setup randomisations mentioned in Section 4.3.2, we randomise the environment parameters, introduce external disturbance and add uniformly sampled observation noise such that RMA can adapt to diverse scenarios. These are summarised in Table 1, where the parameters start at their default values, and the randomisation ranges are linearly ramped towards low and high values (see Algorithm 1 and 2). Each bimanual task has tailored values, but as an example we found that, for the Lift YCB task, the agents perform well with $\text{step}_{\text{rand\_start}} = 30,000,000$ and $\text{step}_{\text{rand\_end}} = 35,000,000$.

Table 1: Domain randomisation ranges for environment parameters.

Rand Type	**Parameter**	**Default**	**Low**	**High**
Env. Parameter	Object Scale	1.0	0.70	1.20
	Object Density	1.0	0.50	5.00
	Object Friction	1.0	0.50	1.10
External disturbance	Force	0	0	2.0
Observation Noise	Joint Position Noise	0	-0.005	0.005
	Object Position Noise	0	-0.005	0.005

4.3.5 Training and Model Architecture Details

During base policy training, for Pick YCB task, we train for 150M steps and 70M global steps for the other two tasks. We utilise a batch size of 12,800 or 25,600 (i.e. number of training environments $\times$ number of steps in each episode) where the mini-batch size is 32 for the PPO updates. We set the PPO clip coefficient ϵ to 0.2, learning rate to 0.0003, discount factor γ to 0.8, and we use the Adam optimiser [47]. For adaptation training, we train for 2M steps, with the

Adam optimiser minimising the MSE loss. Lastly, we describe the model architectures below:

- **Actor Network:** A 4-layer MLP with hidden dimensions of 256 and Tanh activations. It outputs the mean of a Gaussian action distribution. A learnable log_std parameter defines the standard deviation.
- **Critic Network:** A 4-layer MLP with 256 hidden units and Tanh activations, terminating with a scalar value output.
- **Environment encoder:** A 3-layer MLP with hidden size 128 and ELU activations, each followed by LayerNorm.
- **Adapter Network:** A 2-layer MLP (with ReLU) which merges proprioception and depth outputs into an environment embedding.
- **Proprioception CNN:** First applies a 2-layer fully connected (FC) MLP with LayerNorm and ReLU, applied equally and independently to each time step (i.e. shared MLP across time). Then uses a 4-layer 1D CNN with progressively smaller kernels (9, 7, 5, 3) and ReLU activations for temporal feature extraction.
- **Depth CNN:** A 3-layer 2D CNN with increasing channel sizes (32, 64, 128), ReLU, BatchNorm, and max pooling, followed by 2 FC layers projecting to the final feature dimension.

4.3.6 Hardware Details

We use a variety of GPUs (Tesla P40, NVIDIA Quadro RTX 6000 and NVIDIA Quadro RTX 8000) from the University of Oxford, Visual Geometry Group (VGG) compute cluster to carry out training. With the RTX 6000, we report approximately 2200 frames per second (FPS) in simulation rollouts. Experiments are executed on the cluster using the SLURM workload manager.

4.4 Implementation Details

Our Python implementation of bimanual RMA in ManiSkill 3 can be found here: `https://github.com/seungbinjoo/bi_rma`, under the `bi_rma/rma/` and `bi_rma/task/` files. The repository includes PyTorch implementations of the RMA pipeline, environment wrappers, training and

evaluation scripts, detailed hyperparameter configurations and numerous implementation details that were essential to producing the working pipeline.

4.4.1 Software Architecture

Our implementation is summarised below in Figure 5. `base_policy_ppo.py`, `adaptation.py` and `evaluation.py` are Python scripts that run the base policy training, adaptation training, and evaluation (deployment in simulation), respectively.

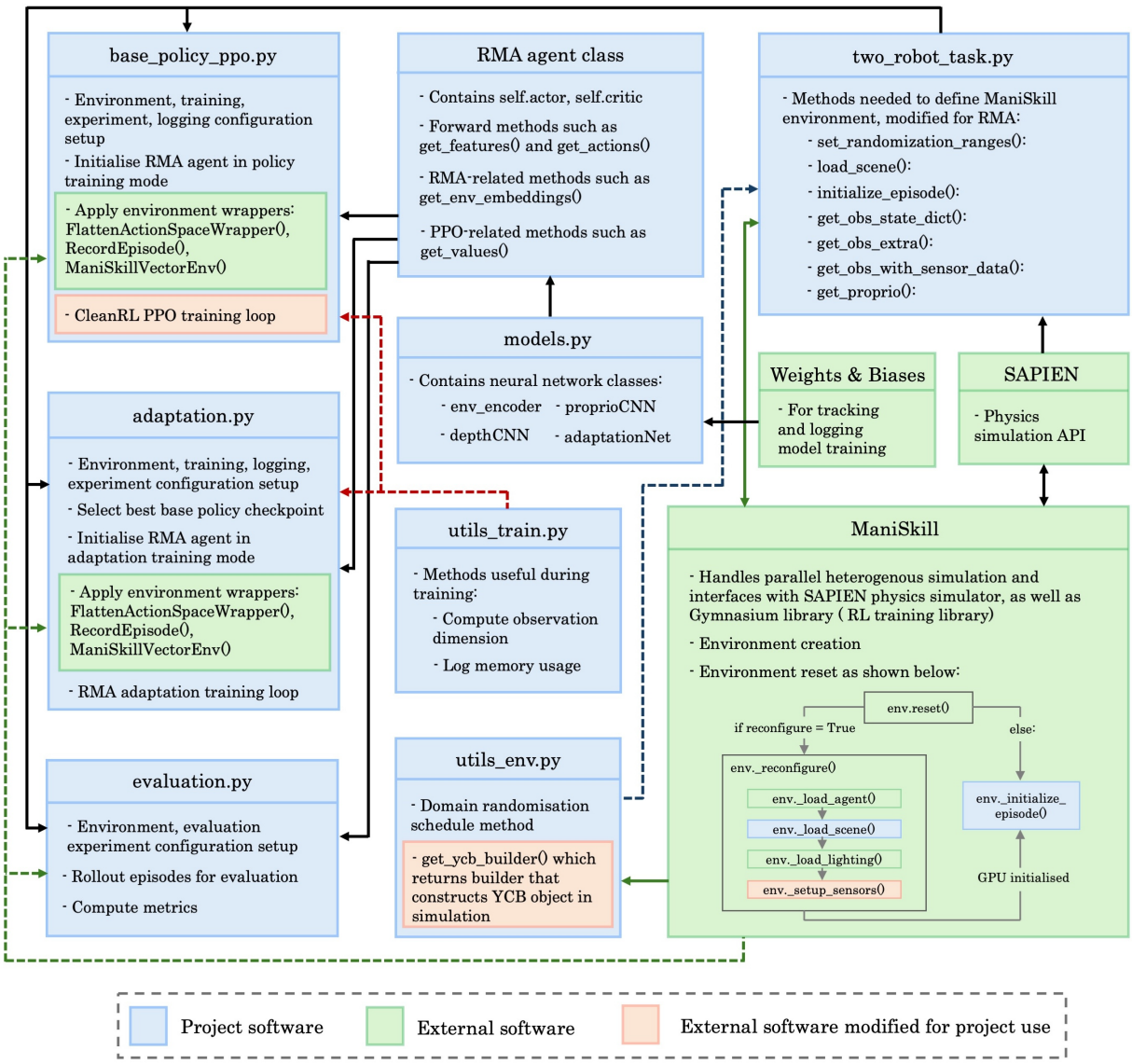

Figure 5: Software architecture diagram.

All three scripts use the RMA agent class which contains the actor critic networks and RMA-

related methods such as `get_env_embeddings()`. The RMA agent class accesses `models.py` to obtain classes that specify all PyTorch [48] neural network modules with architectures described in Section 4.3.5. The models' performances and metrics are tracked via Weights and Biases. The three scripts also draw from `two_robot_task.py`, which are the task environments built using the ManiSkill and SAPIEN API and adapted to pass observations and track proprioception history such that RMA is possible. Finally, `utils_train.py` and `utils_env.py` contain helper methods that are utilised in the training and environment building code, respectively.

4.4.2 Environment Reconfiguration Frequency

We refer the reader to the ManiSkill code flow diagram in Figure 5 (bottom-right, green box), which illustrates the environment reset mechanism in ManiSkill 3. When `env.reset()` is called (e.g. after the end of an episode in a parallel environment), ManiSkill checks whether the `reconfigure` flag is `True`. If so, this triggers the reconfiguration process via `env._reconfigure()`, which then loads the robot arms (`env._load_agent()`), the table, object and goal position indicator (`env._load_scene()`), the scene's lighting (`env._load_lighting()`) and the wrist-mounted depth sensors (`env._setup_sensors()`) into the simulation, ensuring all initial placements avoid collisions. Afterwards, `env._initialize_epsisode()` sets and randomises the positions of the agents and objects in the scene such that an episode can begin. Typically, full reconfiguration is only performed once at the start of training; subsequent calls to `env.reset()` set `reconfigure` to `False`, triggering only `env._initialize_episode()` to reset positions while keeping the scene structure and loaded entities fixed. In our case, however, the object identity, scale and friction must be randomised per episode—logic implemented inside `env._load_scene()`. Thus, we set the reconfiguration frequency to 1, forcing a reconfiguration every `env.reset()`. While this introduces simulation overhead, we observe that performance remains satisfactory (see Section 4.3.6).

4.5 Experimental Results

In Figure 6, we show the learning curves for bimanual RMA base policy training (top plots) and adaptation training (bottom plots) for Two Robot Pick YCB, Lift YCB and Stack Cube tasks. For

base policy training, we report two evaluation metrics: **success_once**, defined as the proportion of 512 evaluation environments in which the bimanual agent satisfies the success condition at least once during an episode; and **reward**, the average reward across all evaluation environments.

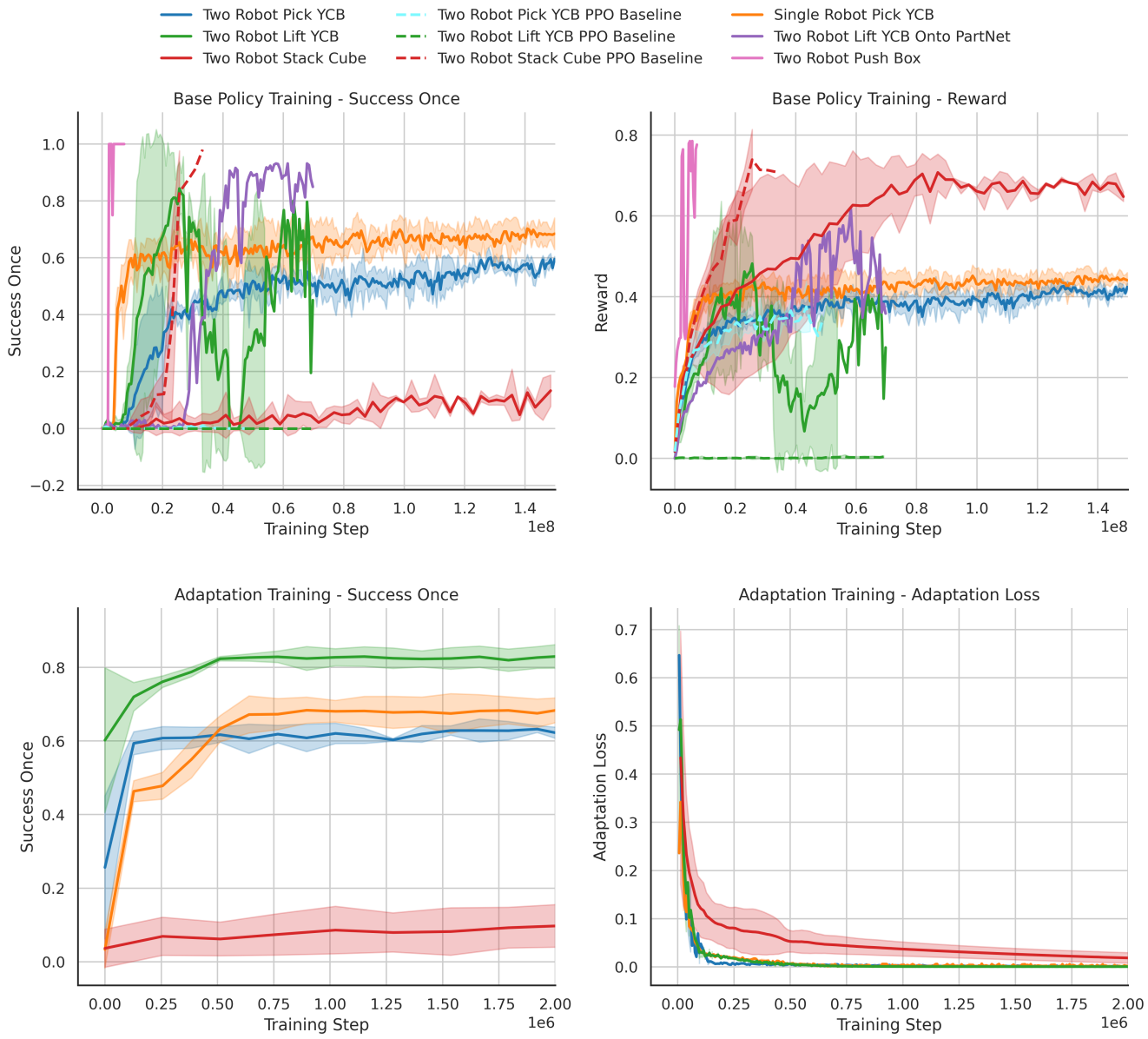

Figure 6: The top two plots show learning throughout base policy training, top left showing success once rate and top right showing mean normalised reward. Bottom two plots track adaptation training, through success once rate, as well as adaptation loss (i.e. MSE between ground truth and predicted environment embedding). Each line represents the mean of 4 runs with random seeds (9351, 4796, 1788, 6005) and the shaded area represents the standard deviation across the 4 runs. We plot the main bimanual RMA tasks in red, green, blue, and their corresponding PPO baselines in dotted lines. We also plot Single Robot Pick YCB (for comparison reasons), and results for Two Robot Lift Onto PartNet and Push Box tasks, which are relevant later for Section 5.4.1.

In both evaluation metrics, the bimanual RMA base policy demonstrates strong performance on the Two Robot Pick YCB task (blue line), with the success_once metric converging to approximately 60%. We compare this with the RMA base policy trained on the Single Robot Pick YCB task (orange line), a simplified variant that involves a single robot performing the pick-and-place operation without requiring a bimanual handover. Although Two Robot Pick YCB presents a significantly more complex and long-horizon challenge, the bimanual RMA policy achieves a success_once rate within 10% of the single-robot counterpart. We further compare this to a PPO baseline (cyan dotted line) trained on the same Two Robot Pick YCB task under identical environment randomizations, but without privileged environment information or the environment encoder. This ablation highlights the importance of these components, as the success_once rate drops sharply to nearly 0% during training when they are removed.

Similarly, the bimanual RMA for the Two Robot Lift task (green line) performs well for both metrics, reaching up to approximately 90% success_once rate. In contrast, the corresponding PPO baseline (green dotted line) fails to make progress, maintaining a success_once rate of 0%. We observe notable fluctuations in both success_once and reward during training, which we attribute to the task's sensitivity: minor suboptimal policy updates can lead to failure cases where the robots drop the object during lifting, resulting in a significant drop in reward. We hypothesise that experimenting further with the PPO clip parameter (which controls the size of policy updates) could potentially help stabilise learning.

For the Two Robot Stack Cube task (red line), bimanual RMA base policy showed relatively poor performance, with success_once staying below 20%, even after 150M steps. The PPO baseline (red dotted line), however, showed great progress (above 90% success rate), significantly outperforming bimanual RMA base policy in both metrics. This discrepancy suggests that bimanual RMA base policy delivers the most benefit when the privileged information is rich and high-dimensional—e.g. detailed YCB object geometry—because the environment encoder can exploit that complexity. In the Stack Cube task, the privileged information is relatively low-dimensional (friction, scale, density); PPO can absorb these variations through ordinary domain randomisation, and the extra RMA environment encoder merely adds computational overhead and slows learning.

Figure 6 shows bimanual RMA base policy applied to Two Robot Lift YCB Onto PartNet and Push Box tasks. This is relevant in Section 5.4.1 and can be disregarded for the current analysis.

To begin adaptation training (bottom plots in Figure 6), we select the bimanual RMA base policy checkpoint with the highest success_once after surpassing $\text{step}_{\text{rand\_end}}$ in each run. This checkpoint is then used to execute adaptation training, during which we track both success_once and the adaptation loss (MSE between predicted and ground truth environment embeddings). In all cases, success_once converges within 2M steps to levels comparable to the corresponding base policy. Additionally, the adaptation loss drops rapidly across tasks, indicating that the proprioception CNN, depth CNN, and adapter network can effectively learn the mapping to ground truth embeddings. These results suggest that performance is primarily bottlenecked by the base policy training—since the final adaptation policy can only perform as well as the privileged-information-trained base policy it builds upon.

We extract the best performing checkpoints from Figure 6 and show the results in Table 2.

Table 2: Bimanual RMA results and performance comparison with other variants, methods and benchmarks. We show mean success_once rates and standard deviations. Base Policy (PPO) is bimanual RMA base policy optimised using PPO as discussed before. PerAct2 [25] is a state-of-the-art IL bimanual manipulation benchmark. The PerAct2 Two Robot Pick task only involves a cube object (instead of a randomly sampled YCB object) and thus is an easier version of the Two Robot Pick YCB task. PPO is the baseline that uses no privileged information and environment encoder. uni-RMA (PPO) is the RMA pipeline applied to single robot arms. bi-RMA (SAC) is the bimanual RMA pipeline exactly as discussed before but optimised using SAC. This is discussed later in Section 5.1.5 and can be disregarded as of now. Finally bi-RMA (PPO) is the bimanual RMA pipeline performance with the trained base policy and adapters at deployment phase.

	Two Robot Pick YCB	Two Robot Lift YCB	Two Robot Stack Cube
Base Policy (PPO)	*66.17 ± 0.74*	*87.6 ± 5.40*	*15.23 ± 6.78*
PerAct2	41.00*	50.00	—
PPO	1.16 ± 0.98	0.00 ± 0.00	94.57 ± 1.22
uni-RMA (PPO)	71.35 ± 3.30 *	0.00 ± 0.00	0.00 ± 0.00
bi-RMA (SAC)	—	51.75 ± 8.38	—
bi-RMA (PPO)	**65.74 ± 0.88**	**85.13 ± 4.58**	**13.69 ± 5.54**

Table 2 clearly shows that our bimanual RMA pipeline performs well with generalisable bimanual manipulation tasks, namely, the Two Robot Pick YCB and Two Robot Lift YCB tasks. For these tasks, our method outperforms PerAct2, a state-of-the-art bimanual manipulation IL method.

With the exception of the Two Robot Stack Cube task, our bimanual RMA policy significantly outperforms PPO, what is widely considered the state-of-the-art standard RL algorithm.

In Figure 7, we present snapshots across successful episodes for the 3 bimanual tasks.

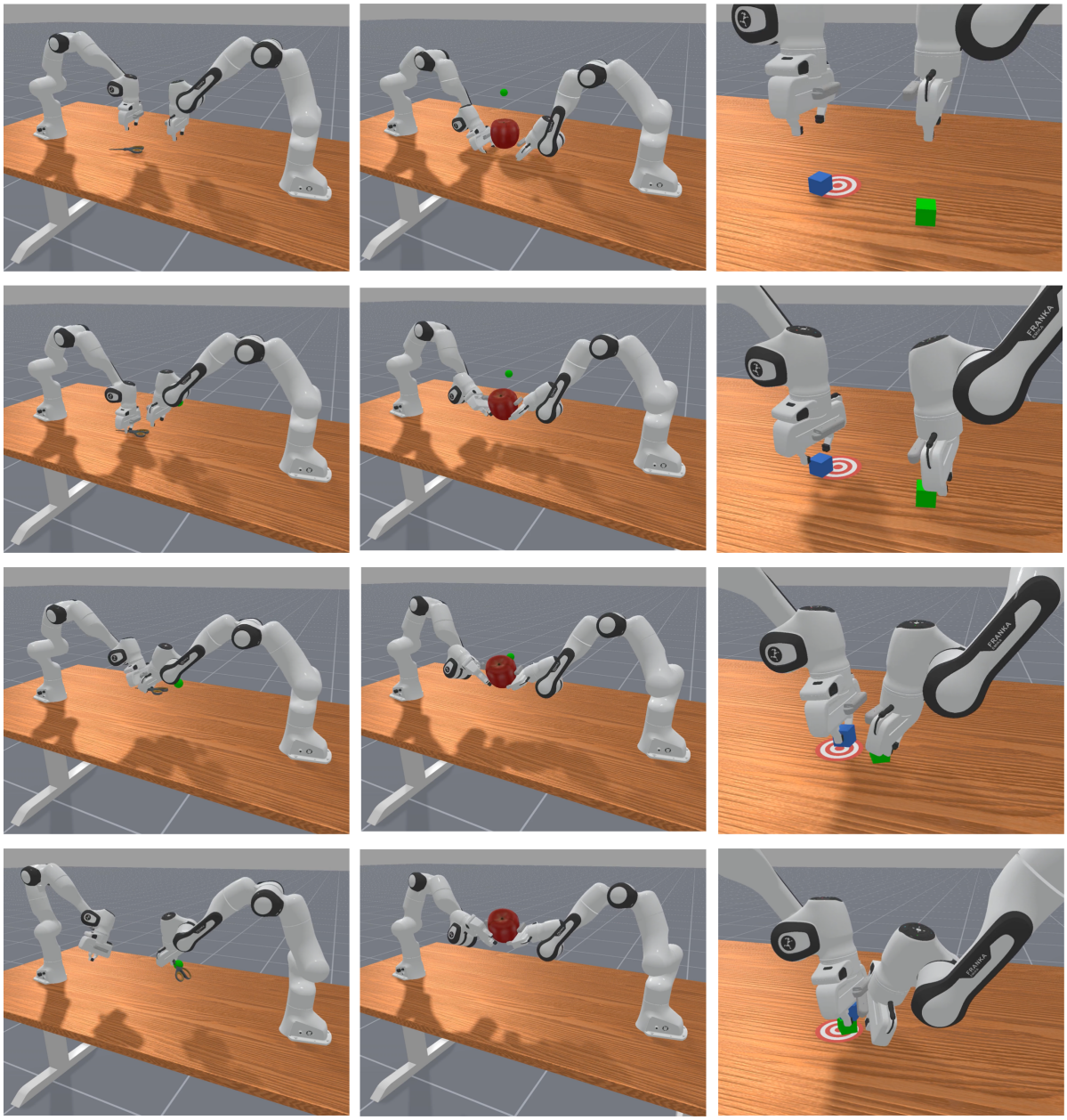

Figure 7: Snapshots from successful episodes from each bimanual task: Two Robot Pick YCB (left column), Two Robot Lift YCB (middle column), Two Robot Stack Cube (right column). For the Two Robot Pick YCB task, the object being manipulated is a pair of scissors sampled from the YCB dataset. For the Two Robot Lift task, an apple is being manipulated. We draw attention to the interesting behaviour where the robots learn to rotate their end effectors anti-symmetrically to help balance the round object to be lifted. Finally, in the Two Robot Stack Cube task, the robots manipulate cubes with randomised physical parameters, such as density and friction.

Overall, the strong quantitative and qualitative results demonstrate that our bimanual RMA pipeline effectively addresses generalizable bimanual manipulation tasks requiring adaptation to diverse environment configurations and objects.

5 Reinforcement Learning with VLM Feedback

From Section 4, we have developed a generalisable bimanual manipulation RL policy—bimanual RMA—capable of adapting to diverse objects and environment parameters. However, in its current form, the policy relies on user-specified goal states to define the task objective—for instance, specifying a target position vector for the Two Robot Pick Single YCB task. This approach lacks semantic understanding: the policy treats objects purely as entities to be moved to designated coordinates, without reasoning about their meaning, category, or functional relevance. As it stands, our policy alone also cannot use text-based task descriptions and has no vision-language understanding as motivated in Section 1.1. Lastly, we note that, currently, our policy aims to maximise the cumulative rewards given by a dense reward function which is handcrafted. As discussed in Section 2.3, and demonstrated with Algorithm 3 (which required a significant reward engineering effort to reach its final form as presented), designing such dense reward functions requires expertise and extensive amounts of trial-and-error, which is neither scalable nor practical for a broad range of tasks.

Therefore, in this section, we aim to imbue vision-language understanding into the pipeline by learning the reward model using vision-language model (VLM) feedback. To that end, we first discuss PEBBLE, a method for efficient feedback-based RL.

5.1 Feedback-Efficient Interactive Reinforcement Learning via Relabeling Experience and Unsupervised Pre-training (PEBBLE)

Unsupervised **Pr**-**E**training and preference-**B**ased learning via rela**B**e**L**ing **E**xperience (PEBBLE) is a method for achieving efficient feedback-based off-policy RL [49]. At its core, PEBBLE is based on preference-based RL which we introduce below.

5.1.1 Preference-based RL

In preference-based RL, we learn a reward function $\hat{r}_\psi$ (parametrised as an MLP with weights ψ) using preferences from a teacher (e.g. human). The teacher provides preference labels over the agent's behaviours and the reward model is updated to align with the teacher's preferences.

We denote a sequence of agent observations and actions $\{x_t, a_t, x_{t+1}, a_{t+1}, ..., x_{t+k}, a_{t_k}\}$ as a segment σ. We define y as the teacher preference between two segments σ^0 and σ^1, where $y = 0$ and $y = 1$ indicates a preference for σ^0 and σ^1, respectively, and $y = -1$ indicates equal preferability between the segments. Then, given the parametrised reward function $\hat{r}_\psi$ and two segments, we can use the Bradley-Terry model [50] to compute the probability that the reward function $\hat{r}_\psi$ prefers segment σ^1 over segment σ^0:

$$P_\psi[\sigma^1 \succ \sigma^0] = \frac{\exp \sum_t \hat{r}_\psi(\boldsymbol{x}_t^1, \boldsymbol{a}_t^1)}{\sum_{i\in\{0,1\}} \exp \sum_t \hat{r}_\psi(\boldsymbol{x}_t^i, \boldsymbol{a}_t^i)} \tag{10}$$

This means that given a dataset of segment pairs and a teacher preference $D = (\sigma_i^0, \sigma_i^1, y_i)$, we can learn a reward function to align with teacher preferences by minimising the following loss [36]:

$$\begin{aligned} \mathcal{L}_{\text{Reward}} = -\mathbb{E}_{(\sigma^0,\sigma^1,y)\sim\mathcal{D}}[&\mathbb{I}\{y = (\sigma^0 \succ \sigma^1)\} \log P_\psi[\sigma^0 \succ \sigma^1] \\ &+\mathbb{I}\{y = (\sigma^1 \succ \sigma^0)\} \log P_\psi[\sigma^1 \succ \sigma^0]] \end{aligned} \tag{11}$$

where $\mathbb{I}$ is an indicator function which equals 1 when the condition inside is true, and 0 otherwise. The above loss is similar in logic to binary cross entropy loss.

As illustrated in Figure 8, in preference-based RL, the standard paradigm is to first allow the policy π to interact with the environment and update its parameters according to an RL algorithm such as PPO or SAC. Then, segments are sampled from the trajectories collected and we *query* the teacher with q_i for preference labels y_i. The reward function is subsequently updated by performing supervised learning with respect to the above loss. This process continues by alternating between policy optimisation and reward function learning. We note that policy learning and reward learning processes are decoupled and thus can run asynchronously [51]. Lastly, as is typically done in preference-based RL works [36], an *ensemble* of reward predictors are learned. The actual reward

estimate $\hat{r}$ is given by independently normalising the reward from each predictor and computing the mean of the normalised rewards across the ensemble.

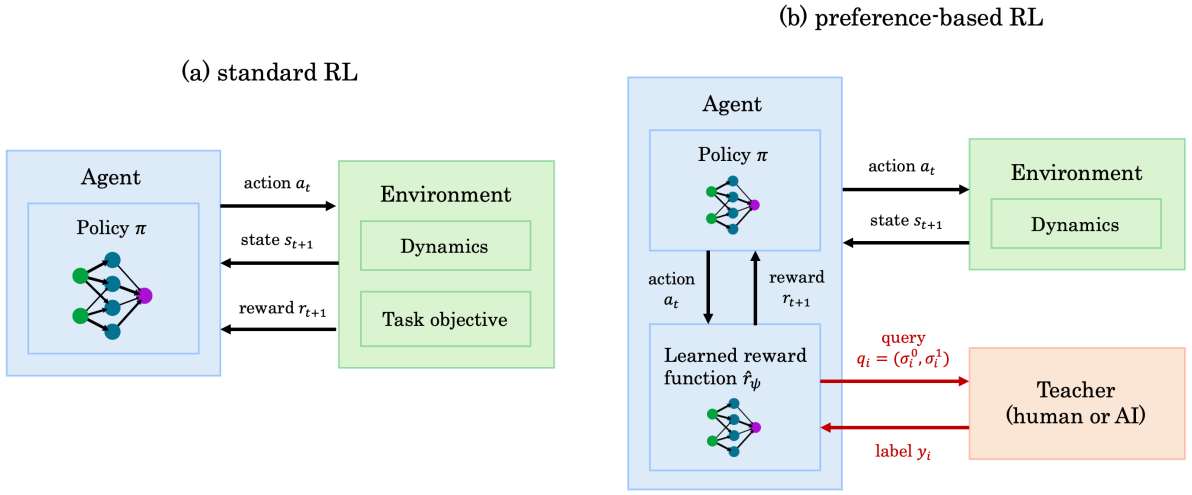

Figure 8: Diagrams of standard RL (a) and preference-based RL (b). In standard RL, the agent interacts with the environment via actions and receives rewards from the environment. In preference-based RL, the agent instead receives rewards from a learned function. The core idea is that the reward function $\hat{r}_\psi$ is learned from preference labels over segments.

5.1.2 Unsupervised Pre-training of the Policy with State Entropy Reward

In the early stages of RL, the agent typically follows a near-random policy that lacks goal-directed behavior. As a result, its actions are often erratic and fail to explore the state space effectively. At this point in training, it is challenging to obtain meaningful preference queries, as a teacher cannot reliably express preferences between two incoherent trajectory segments. Consequently, the early stages of training can suffer from sample inefficiency; preference-based RL can require many queries before substantive progress begins to show.

PEBBLE addresses this issue by pre-training the policy in an unsupervised manner (i.e. using data with no labels) by using an intrinsic reward function that promotes the exploration of diverse states [49], [52]. To that end, PEBBLE utilises the state entropy $\mathcal{H}(s)$ as an intrinsic reward. In information theory, entropy quantifies uncertainty or randomness in a probability distribution, expressing how unpredictable or diverse the outcomes are. In our case, we define state entropy as:

$$\mathcal{H}(\mathbf{s}) = -\mathbb{E}_{\mathbf{s}\sim p(\mathbf{s})}[\log p(\mathbf{s})] \tag{12}$$

which measures the diversity of visited states under the policy. By pre-training the policy to maximise this cumulative state entropy reward, the policy learns to generate diverse and more distinct behaviours which form more informative queries that accelerate the preference-based reward learning process in the beginning stages of training.

In many settings, however, the true distribution over states visited by the policy $p(s)$ is unknown, and the state space being continuous and high-dimensional makes it computationally intractable to get expectations over the states. One way to overcome this is to use a particle-based entropy estimator $\hat{\mathcal{H}}(\mathbf{s})$ [49], [53], [54]:

$$\mathcal{H}(\mathbf{s}) \propto \sum_i \log(||\mathbf{s}_i - \mathbf{s}_i^k||) \tag{13}$$

Here, $\mathbf{s}_i^k$ is the k-th nearest neighbour (kNN) of $\mathbf{s}_i$. Equation 13 implies that we can increase state entropy by increasing the distance between a state and its kNN. Using this estimator, PEBBLE defines the following intrinsic reward $r^{\text{intrinsic}}$ to guide unsupervised pre-training:

$$r^{\text{intrinsic}}(\mathbf{s}_t) = \log(||\mathbf{s}_t - \mathbf{s}_t^k||) \tag{14}$$

where $||\mathbf{s}_t - \mathbf{s}_t^k||$ is the Euclidean distance between the current state and its k-nearest neighbour from a replay buffer of past states. During the unsupervised pre-training phase, we use SAC (see Section 3.2.2) to maximise the intrinsic reward $r^{\text{intrinsic}}$, thereby encouraging the agent to explore a more diverse range of states before any teacher feedback is introduced.

5.1.3 Sampling Strategy for Maximizing Expected Value of Information in Queries

In the preference-based RL setup, an ideal query selection strategy aims to maximise the expected value of information (EVOI)—that is, to select segment pairs whose associated preferences are expected to yield the greatest improvement in the agent's performance when used to update the reward function [49], [55]. Computing EVOI requires an expectation over all possible trajectories, which in robot manipulation contexts, is intractable. Thus, many prior works, including PEBBLE, explore approximate strategies for selecting informative queries that are likely to effectively improve

the reward function. We outline two such sampling approaches below:

- **Uniform sampling:** As the name implies, we uniformly and randomly sample segment pairs from the trajectories collected by the policy.
- **Ensemble disagreement-based sampling:** A large set of segment pairs of length k is sampled from the agent's most recent interactions with the environment. Each reward model in the ensemble is used to predict the reward for the sampled segment pairs. The segment pairs exhibiting the highest variance in predicted reward across the ensemble members are selected as informative queries for preference-based RL [36].

Prior works show that ensemble disagreement-based sampling is slightly superior to uniform sampling, including in robot manipulation tasks [49]. However, due to time constraints and for implementation simplicity, we employ uniform sampling in this work and leave exploring ensemble disagreement-based sampling as future work.

5.1.4 Efficient Feedback with Off-policy RL

In preference-based RL, training alternates between learning the reward function and optimizing the policy. After each reward learning phase, the actor and critic networks are updated using an RL algorithm. However, because the learned reward function $\hat{r}_\psi$ is updated continuously during training, the reward signal is inherently *non-stationary*. This means past transitions may no longer align with the current reward function, leading to inconsistencies between how earlier and current transitions are evaluated.

To handle reward non-stationarity, previous works [36] use on-policy RL algorithms such as trust region policy optimisation (TRPO) [56] and advantage actor-critic (A2C) [57]. This works because, as elaborated in Section 3.2, these on-policy RL methods update the policy using trajectories collected under the current policy and reward function, subsequently discarding previous trajectory data. While this mitigates the mismatch between resulting from a changing reward function, it also leads to poor sample and feedback efficiency.

Instead, PEBBLE uses off-policy methods that store and reuse agent experiences in a replay buffer (e.g. as done in SAC described in Section 3.2.2) for higher sample and feedback efficiency.

The main issue is that the replay buffer contains transitions (s_t, a_t, r_t, s_{t+1}) where the reward r_t is labelled using previous and outdated learned reward models. Updating the policy using such transitions leads to training instability. PEBBLE addresses this issue and achieves stable learning by simply relabelling all of the agent's past experiences in the replay buffer with the most recent reward model, after every reward function update.

In summary, PEBBLE combines unsupervised pre-training phase that promotes diverse state exploration early on and off-policy learning with experience relabelling to achieve feedback efficient preference-based RL.

5.1.5 On-Policy PPO and Preference-based RL

We note that our bimanual RMA pipeline from Section 4 utilises on-policy PPO. In Section 4.5 and Table 2, we therefore included a new variant of bimanual RMA that was implemented using off-policy SAC instead. This led to a drop in performance (success_once from 85.13% to 51.75% for the Two Robot Lift YCB task). However, we still opt for using bimanual RMA with SAC, because, based on prior works such as *B-Pref* [58] which provide benchmarks on various preference-based RL approaches, *PrefPPO* (standard preference-based RL described in Section 5.1.1 fused with PPO) is heavily outperformed by off-policy PEBBLE, particularly in robot manipulation tasks. Moreover, whether the teacher is a human or AI labeller (as in our case), feedback is expensive and we therefore prioritise feedback efficiency. Therefore, our approach outlined in the next sections will use bimanual RMA based on SAC in conjunction with PEBBLE.

5.2 Reinforcement Learning from Vision-language Foundation Model Feedback (RL-VLM-F)

Reinforcement learning from vision-language foundation model feedback (RL-VLM-F) [59] builds upon PEBBLE by using a vision-language model (VLM) to generate preference labels (instead of a human labeller as done in PEBBLE). A VLM is a large pre-trained machine learning model that can process both text and images and generate outputs that rely on their multi-modal relationship. Thus, RL-VLM-F uses image observations (snapshots of a single time step) as seg-

ments σ, as opposed segments defined by a sequence of agent observations and actions as described in Section 5.1.1. As explained previously, after a policy optimisation phase, policy rollouts are executed to collect trajectories. Then, RL-VLM-F samples observation pairs from those trajectories and subsequently queries a VLM for a preference over two image observations. The VLM provides a preference, which we use to learn the reward function as described in Section 5.1.1, 5.1.2 and 5.1.4 with preference-based RL and PEBBLE.

5.3 Bi-GRASP: Combining RMA and RL-VLM-F

In this section, we introduce our pipeline, **Bi**manual **G**ene**R**alisable manipulation with **A**daptation and vi**S**ion-language **P**references (Bi-GRASP), which simultaneously tackles the 3 project objectives set out in Section 1.1. Our Bi-GRASP method combines RMA [20] and RL-VLM-F [59] to achieve generalisable bimanual manipulation agents that solve text-described tasks requiring vision-language understanding.

5.3.1 Bi-GRASP Algorithm Overview

Bi-GRASP extends the RL-VLM-F framework through several key enhancements. Firstly, Bi-GRASP achieves generalisable manipulation of diverse objects by integrating RMA into the RL-VLM-F pipeline. As demonstrated in Section 4.1, we note that in our bimanual RMA pipeline, the base policy training and adaptation training are decoupled. Similarly, RL-VLM-F—like many preference-based RL methods—separates policy optimisation from reward learning. This decoupled structure enables us to incorporate RMA's base policy and environment encoder training into the policy optimisation phase of RL-VLM-F. We then alternate between reward learning and policy optimisation, before concluding with the adaptation training phase entirely after the reward model is learned. Secondly, Bi-GRASP focuses on bimanual manipulation tasks, which contrasts with the relatively simple and short-horizon single-arm manipulation tasks explored by RL-VLM-F [59]. Lastly, Bi-GRASP uses a novel off-policy variant of RMA based on SAC, enabling feedback-efficient learning through the PEBBLE algorithm, while also maintaining RMA's adaptive behaviour. The full Bi-GRASP pipeline is visualised in Figure 9.

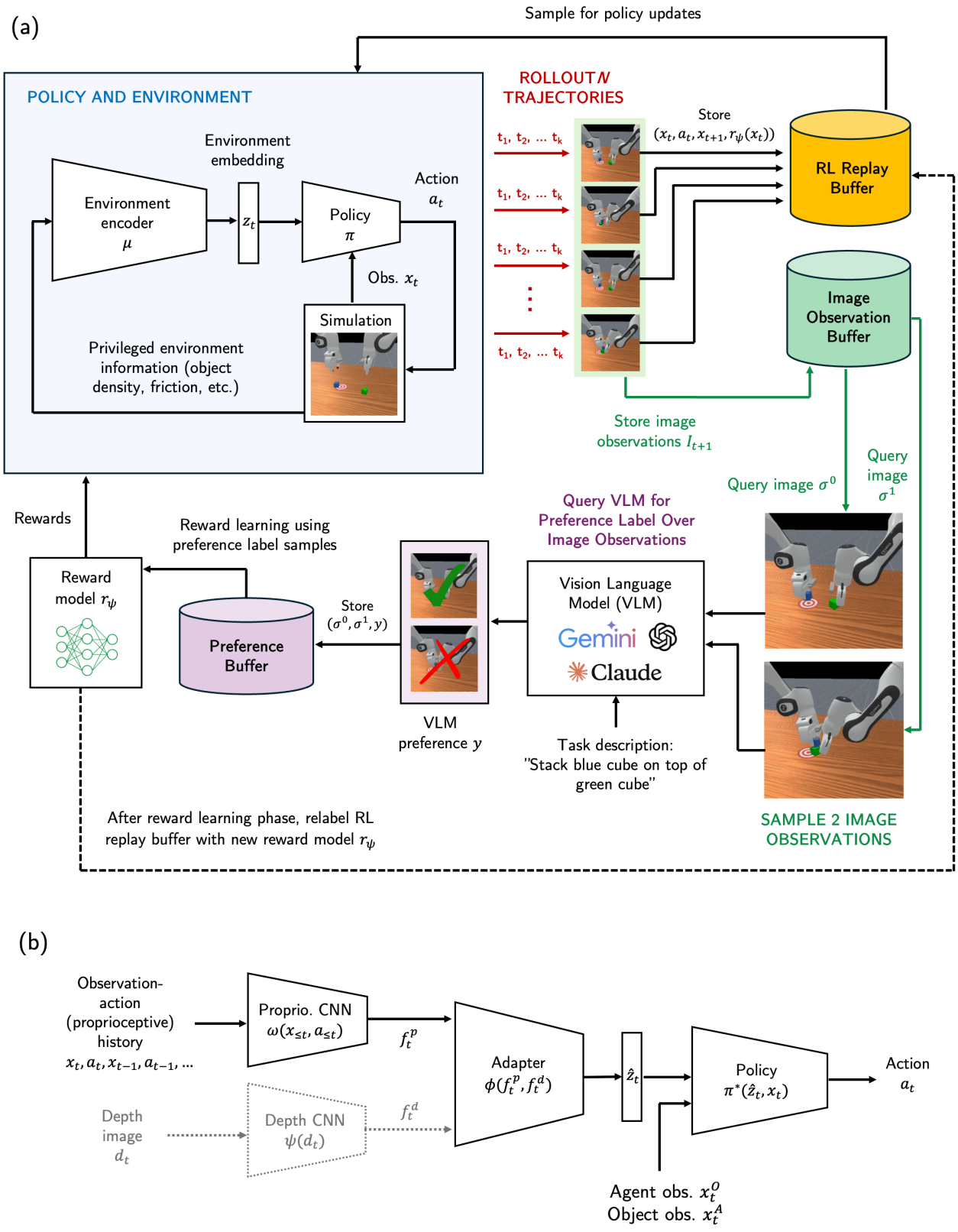

Figure 9: Bi-GRASP: (a) Policy optimisation and reward learning process based on PEBBLE and RL-VLM-F. We fuse RMA into this automatic reward learning pipeline by having the policy learn using privileged environment information. (b) As in standard RMA, an adapter is learned, but after the policy and reward learning phases. The depth CNN is greyed out as it is optional for some of the bimanual tasks we show in Section 5.4.1.

Bi-GRASP training is split into two main phases. In the first phase (Figure 9(a)), we begin by allowing the policy π and environment encoder μ to interact with the environment. At this initial stage, the policy π executes unsupervised exploration, updating itself using SAC to maximise the PEBBLE intrinsic reward (see Section 5.1.2). Afterwards, we proceed to alternating stages of policy learning and reward learning. Every iteration, we use current policy π and environment encoder μ to rollout N trajectories. The transitions $(x_t, a_t, x_{t+1}, r_\psi(x_t))$ are stored into an RL replay buffer, and the image observations from those trajectories are stored into an image observation buffer. A batch of transitions are sampled from the RL replay buffer to jointly update the policy π and environment encoder μ using SAC. Subsequently, we begin reward learning by sampling queries, i.e. pairs of images (σ^0, σ^1), from the image observation buffer (using ensemble disagreement-based sampling). We send each image pair and the text description of the task to a VLM, requesting a preference label y. Segment pair and label tuples (σ^0, σ^1, y) are stored in a preference buffer. We sample (σ^0, σ^1, y) data from the preference buffer and learn the reward model r_ψ by minimising Equation 11. Finally, the updated reward model r_ψ is used to relabel the RL replay buffer (PEBBLE's relabelling technique).

After policy optimisation and automatic reward learning, we execute the second training phase (Figure 9(b)), which is essentially the adaptation training from bimanual RMA. Once again, the adapter models are learned via supervised learning with the predicted $\hat{z}_t$ and ground truth z_t.

Below in Algorithm 4, we present the Bi-GRASP algorithm in formal notation. We note Algorithm 4 heavily builds upon RL-VLM-F and PEBBLE algorithms, incorporating key modifications to support generalisable bimanual manipulation. Importantly, Bi-GRASP utilises RL-VLM-F to automatically learn reward functions with VLMs, which avoids the expensive process of handcrafting rewards such as those shown in Algorithm 3.

5.3.2 VLM Query Process

The VLM query process we use is a modified version of the RL-VLM-F query process that we empirically found works well for our experiments. It is shown in Figure 10. Our VLM query is composed of 2 stages. In stage 1, the 2 image observations and the task description are passed,

Algorithm 4 Bi-GRASP (based on RL-VLM-F [59] and PEBBLE [49])

input Text-based task goal g
Initialize policy π_θ, environment encoder μ_θ and reward model r_ψ
Initialize empty preference buffer D, empty RL replay buffer B, empty image observation buffer I, number of policy update steps N_π, number of reward model update steps N_r, VLM query frequency in terms of iterations K, number of VLM queries for each reward learning phase M
// UNSUPERVISED EXPLORATION (Section 5.1.2)
$B, \pi_\theta \leftarrow$ `unsupervised_pretraining()`
// BASE POLICY TRAINING (Section 4.1)
for each iteration *iter* **do**
  // TRAJECTORY DATA COLLECTION
  for $t = 1$ to T **do**
    Collect observations x_{t+1}, image I_{t+1} by executing bimanual RMA base policy $a_t \sim \pi_\theta(x_t, z_t)$ and environment encoder $z_t = \mu(e, s_t)$
    Add transition $B \leftarrow B \cup \{(x_t, a_t, x_{t+1}, r_\psi(x_t))\}$
    Add image observation $I \leftarrow I \cup \{I_{t+1}\}$
  end for
  // RMA POLICY OPTIMISATION (Section 4.1.1 and 3.2.2)
  for $n = 1$ to N_π **do**
    Sample random batch $\{(x_t, a_t, x_{t+1}, r_\psi(x_t))\}_{j=1}^{\text{batch\_size}} \sim B$
    Jointly optimize policy π_θ and environment encoder $\mu(s_t, z_t)$ using the sampled batch with off-policy SAC
  end for
  // PREFERENCE BY VLM (Section 5.3.2 and 5.1.3)
  if iter $\%K == 0$ **then**
    for $m = 1$ to M **do**
      Use uniform sampling to select two image observations (σ^0, σ^1) from buffer I
      Use 2 stage querying with (σ^0, σ^1) and task goal g to generate preference y
      Store in preference buffer $D \leftarrow D \cup \{(o^0, o^1, y)\}$
    end for
    // REWARD LEARNING (Section 5.1.1)
    for $n = 1$ to N_r **do**
      Sample minibatch $\{(\sigma^0, \sigma^1, y)\}_{j=1}^{\text{batch\_size}} \sim D$
      Optimise reward model r_ψ in Equation 11 with respect to parameters ψ
    end for
    Relabel replay buffer B using updated reward r_ψ (Section 5.1.4)
  end if
end for
// ADAPTATION TRAINING (Section 4.1.4)
$\phi^*, \psi^*, \omega^* \leftarrow$ `adaptation_training()` by optimising Equation 9 with supervised learning

and the VLM is asked to directly choose the better image in terms of achieving the task. The stage 1 prompt and VLM response is then inserted at the beginning of the stage 2 prompt. Stage 2 then asks the VLM to respond with a label $y \in \{0, 1, -1\}$, depending on whether it believes the

task goal is better achieved in one of the images, or if there is no difference ($y = -1$). In the latter case, the query images are not used for reward learning.

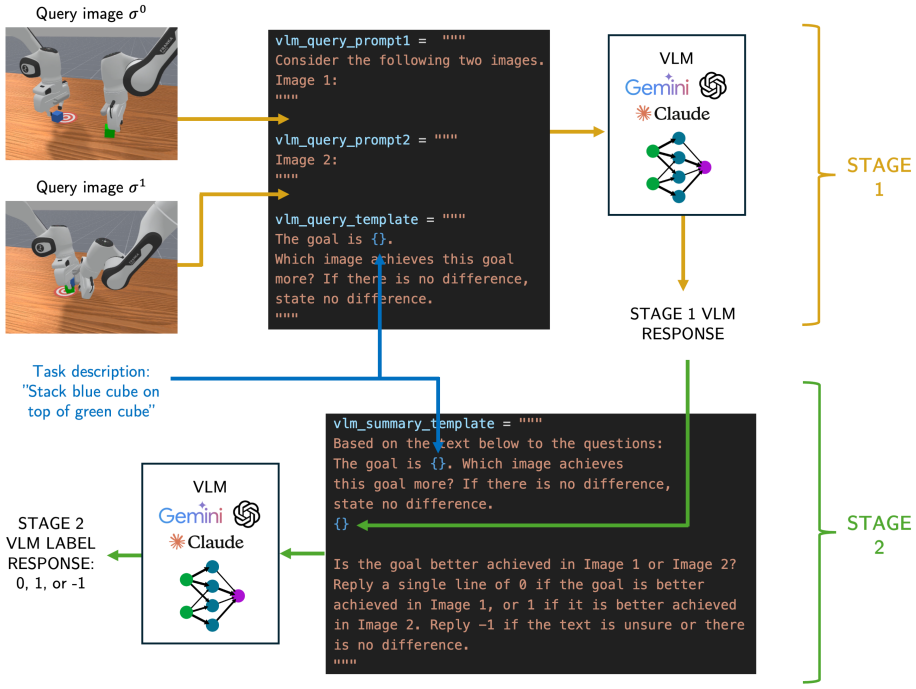

Figure 10: VLM query process to obtain preference label over image observation. This is a similar VLM query strategy and template used in RL-VLM-F [59], except we do not explicitly request for descriptions of each image in stage 1 of the query.

5.3.3 VLM Assumptions and Details

For the RL-VLM-F process in Bi-GRASP to work effectively, our setup must satisfy key assumptions [59]. Firstly, the VLM must be pre-trained on large internet-scale text and image data such that it can reason across diverse and general objects that may be found in our manipulation task environment. This relates to the broad vision-language understanding objective in Section 1.1. Secondly, the VLM must be able to process and compare at least two images concurrently, allowing it to output a preference. Lastly, the progress towards completing a task must be able to be reasonably inferred by the VLM using a single image observation.

We evaluated a range VLMs to identify the one that best satisfied our assumptions. Specifically, we tested Google Gemini 2.5 Flash, 2.0 Flash, 2.5 Pro; Anthropic Claude 3.7 Sonnet; and OpenAI's GPT-4.1, GPT-4.1-mini, and o4-mini. Empirically, we found that OpenAI's o4-mini reasoning

model offered the best trade-off between strong visual reasoning capabilities and cost efficiency, and thus was our final model of choice.

5.4 Experimental Setup

5.4.1 Task Details

Similar to bimanual RMA, Bi-GRASP is also trained and tested using ManiSkill, but with 2 new custom tasks built with the ManiSkill task building API:

1. **Two Robot Lift YCB Onto PartNet:** As shown in Figure 11, this task extends the Two Robot Lift YCB task by requiring the robot to not only lift a sampled round YCB object above a target height, but also to place it on top of a designated drawer, specified via a natural language instruction. The drawer objects are sampled from SAPIEN's PartNet Mobility Dataset. The text-based task description we use is: "lift the round object and place it on top of the white drawer". During training, we randomly sample the round YCB object, while keeping the set of PartNet drawers fixed to simplify the task. If the drawers were also randomly sampled per episode, the policy would need to be explicitly conditioned on the text embedding of the instruction to remain goal-aware—a setting we leave for future work. The dense reward function (used solely for evaluation and not for learning) is adapted from the Two Robot Lift YCB task and extended to provide additional reward for moving the object closer to the top of the target drawer.

2. **Two Robot Push Box:** As illustrated in Figure 11, the task is to push the red box downwards to fully cover the blue rectangle area. The success condition is reached if the red box intersects with more than 85% of the blue area. To enforce bimanual coordination, the robot arms are initialised at a distance such that each can only reach one side of the box. Consequently, a single arm pushing results in tilting and task failure, necessitating simultaneous and coordinated pushing by both arms to achieve success. The dense reward function design involves providing a reward signal based on the area of intersection between the red box and blue area (more details can be found in codebase). The text-based task

prompt we use is "push red box down towards blue rectangle area".

For these two new environments, the environment parameter randomisations for RMA are essentially identical to what was discussed in Section 4.3.4. One significant difference is that the, for the Two Robot Push Box task, we do not vary the box scale or the box identity, and therefore we do not use a depth CNN module during adaptation training.

To verify that these tasks are learnable under the Bi-GRASP framework with learned reward functions, we first conduct a sanity check using the standard bimanual RMA policy introduced in Section 4 and ground truth reward functions. As shown in Figure 6, both the Lift YCB Onto PartNet and Push Box tasks converged relatively quickly, achieving high success rates of approximately 90% and nearly 100%, respectively.

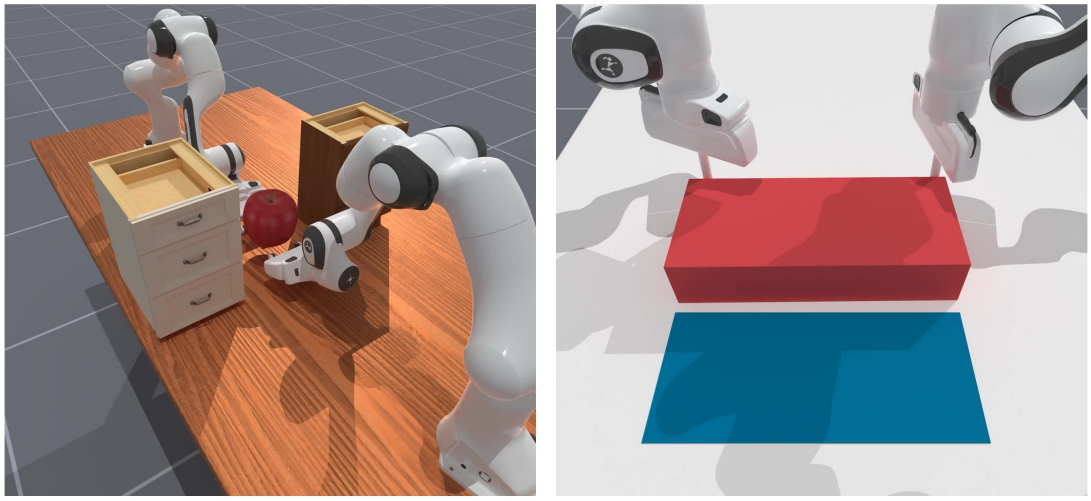

Figure 11: Render of the Two Robot Lift YCB Onto PartNet (left) and Two Robot Push Box (right) task environments.

5.4.2 Simulation Setup Details

For the Two Robot Lift Onto PartNet, the simulation setup in ManiSkill 3 was mostly identical to that of the bimanual RMA pipeline (see Section 4.3.3. For Bi-GRASP, however, it was important to adjust the render camera position and viewing angle such that the robots and objects could be seen by the VLM with minimal occlusions.

For the Two Robot Push Box task, we use a variant of the Panda robot arm, called Panda Stick, which uses a stick as an end effector. For this task, we utilised delta control with respect

to the end effector, so action a_t is the desired change in end effector position (as opposed to joint position). Under this control mode, the end effector rotation is kept constant (which is acceptable as the push box task does not require robots to change their end effector orientations). Once the desired position of the end effector is obtained, the simulation uses inverse kinematics to compute the target joint positions. Lastly, the episode length for this task was set to 16.

For Bi-GRASP experiments, we use NVIDIA RTX A6000 GPUs in the VGG compute cluster.

5.4.3 Model Architecture and Training Details

For the reward models, we use an ensemble size of 3, where each reward predictor is a 3-layer MLP with 128 hidden units and tanh activations. For the bimanual RMA base policy and adaptation models, the architectures are identical to those in Section 4.3.5. We execute 9000 unsupervised pre-training steps, and subsequently alternate between policy optimisation and reward learning every 4000 steps. Query image sizes are 512×512, the batch size for reward learning is 40, and the reward model learning rate is set to 0.0003. Further details on hyperparameters for Bi-GRASP and SAC training can be found in the codebase.

5.5 Implementation Details

The Python code implementation of Bi-GRASP, which integrates the SAC variant of bimanual RMA and RL-VLM-F in ManiSkill 3, can be found here `https://github.com/seungbinjoo/bi_rma`, under the `bi_rma/rl-vlm-f/` file. The Bi-GRASP (and RL-VLM-F code) is built on top of open-source code originally provided by the authors of PEBBLE [49]. Our code makes API requests to the OpenAI developer platform to send and receive messages from o4-mini and OpenAI's other VLMs. The details can be found in the `bi_rma/rl-vlm-f/vlms/` file

5.6 Experimental Results

5.6.1 Effectiveness of Reward Learning

For both the Lift YCB Onto PartNet and Push Box task, the Bi-GRASP pipeline faced many challenges in learning the reward model as well as optimising an effective policy. For instance,

as shown by an evaluation episode in Figure 12, the bimanual agents failed to learn coherent behaviours or make progress towards achieving lifting for the Lift YCB Onto PartNet task. We summarise the main obstacles we observed that prevented effective learning:

1. The primary issue was that even state-of-the-art VLMs exhibit poor spatial reasoning capability and thus often gave completely incorrect feedback. As illustrated in Figure 13, the VLM accuracy—defined as the proportion of feedback instances where the VLM's preferred observation corresponded to a higher ground truth reward—hovered around 0.6 (for Two Robot Push Box task), indicating it was not far from near-randomness.

2. For the Lift YCB Onto PartNet task, VLM queries contained image observations with potentially different YCB objects. VLMs consistently struggled to compare task progress across observations involving different objects—for example, they often preferred images containing a basketball over those with an apple, regardless of the robot's actual state, citing flawed reasoning about the basketball's larger size making it inherently easier to grasp, and thus closer to achieving task success.

3. For the Lift YCB Onto PartNet task, occlusions covering the round YCB object prevented the VLM from giving effective preferences.

4. For the same set of image observations within a query, VLMs often produced highly inconsistent responses due to minor variations in factors such as camera viewpoint or task prompt phrasing (e.g., "push the box to fully cover the blue area" vs. "push the box to minimise the exposed blue area").

5. The high-dimensional action space for these bimanual tasks meant that after the unsupervised pre-training phase, most image observations depicted robot states that were similarly far from task success—often reflecting poor or uncoordinated behaviours across both arms.

To help simplify the task setting, we reduced the action space for the Two Robot Push Box task so that each arm's end effector was constrained to the y-direction. With this simplifcation, we were able to achieve significantly better and successful policy and reward learning performance as shown by the episode in Figure 12.

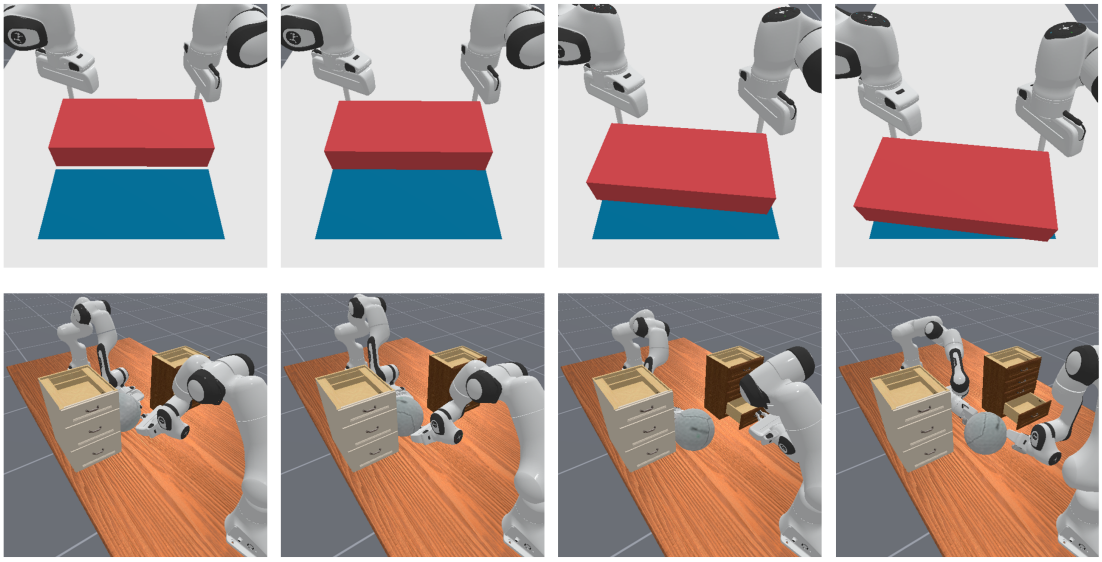

Figure 12: Evaluation episodes for Bi-GRASP in Two Robot Push Box (top) and Lift Onto YCB PartNet (bottom) environments.

5.6.2 Accuracy of VLM Labels and Learned Reward Alignment

In Figure 13, we analyse the VLM label accuracies and the alignment between the learned and ground truth reward models for the y-direction-constrained Two Robot Push Box task.

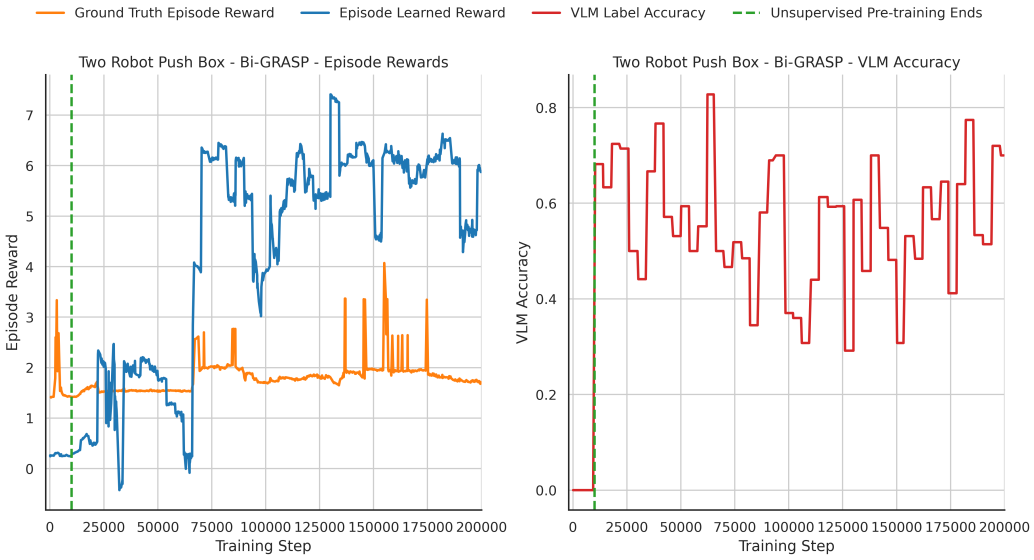

Figure 13: Plot showing episode rewards based on ground truth reward function vs. learned reward function during training (left) and plot illustrating VLM label accuracy during training (right).

As shown on the rightmost plot, the VLM label accuracy was suboptimal, generally hovering around 0.6 and 0.5 during training. Nonetheless, the slight bias toward correct labels was sufficient

to enable meaningful reward learning. Notably, increases in the learned reward often aligned with significant jumps in the ground truth reward (e.g. around step 65k), indicating that the learned reward model captured meaningful aspects of task progress. At 155k training steps, the Bi-GRASP model's RMA base policy reached a 100% success rate across 10 evaluation episodes. Subsequently a 90% success rate was achieved after the adaptation training phase for Bi-GRASP.

Overall, Bi-GRASP demonstrated the possibility of tackling bimanual tasks that require both adaptation to variations in object configuration, as well as vision-language understanding for following a text-based task prompt.

6 Conclusion

6.1 Summary of Contributions

In conclusion, in this report, we introduced bimanual RMA, which allows two arm robot systems to manipulate general objects. We implemented our experiments in a state-of-the-art robot simulator ManiSkill, and showed that our pipeline significantly outperforms modern RL baselines as well as some of the best existing bimanual IL benchmarks. We then built upon bimanual RMA through Bi-GRASP, introducing VLM feedback. This allowed us to specify tasks through text, automatically learn rewards, and distil the broad vision-language understanding of pre-trained VLMs into the agent's learned reward model. Bi-GRASP relies on a novel SAC variant of bimanual RMA and demonstrated good policy and reward learning potential in custom task environments such as Two Robot Push Box. Overall, we have achieved the 3 project objectives set out in Section 1.1 and contributed to research towards more generalist robotic agents.

6.2 Future Work

There is great potential for future work in numerous directions. Most obviously, we could explore deploying the bimanual RMA or Bi-GRASP system in real-life. This would require potentially changing parts of the pipeline to account for the sim-to-real gap (e.g. add another phase of training to RMA that fine-tunes base policy to be robust to inaccurate environment embeddings).

In bimanual RMA, we assume that the object position is passed to the policy as an observation (using an object detector in real-life), but it would be interesting to explore whether this could also be predicted as part of the environment embedding. For example, there could be two separate environment encoders, one for time-invariant environment parameters (e.g. object density) and one for time-variant environment parameters (e.g. object position). There is also much room for developing and testing on more complex and even deformable object bimanual tasks, such as opening a bottle and folding clothes. Moreover, the VLM feedback pipeline and Bi-GRASP system in general has room to improve. One could explore whether it helps to pass privileged information from the simulation as part of the VLM queries so as to help increase VLM label accuracy. A performance drop was caused by using bimanual RMA with SAC, so it would also be interesting to see if bimanual RMA with on-policy PPO could somehow be integrated with RL-VLM-F or PEBBLE. The current Bi-GRASP system does not use a goal-aware policy, so the text task prompt could be transformed into a text embedding and passed as an input to the policy in Bi-GRASP. There is also much potential for experimentation with Bi-GRASP parameters such as the segment size, which could help deal with occlusions during VLM feedback by having multiple observations.

References

[1] V. Mnih *et al.*, *Playing atari with deep reinforcement learning*, 2013. arXiv: 1312.5602 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1312.5602.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, F. Pereira *et al.*, Eds., vol. 25, Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[3] O. Russakovsky *et al.*, *Imagenet large scale visual recognition challenge*, 2015. arXiv: 1409.0575 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1409.0575.

[4] A. Vaswani *et al.*, *Attention is all you need*, 2023. arXiv: 1706.03762 [cs.CL]. [Online]. Available: https://arxiv.org/abs/1706.03762.

[5] D. Silver *et al.*, *Mastering chess and shogi by self-play with a general reinforcement learning algorithm*, 2017. arXiv: 1712.01815 [cs.AI]. [Online]. Available: https://arxiv.org/abs/1712.01815.

[6] B. Calli *et al.*, "Yale-cmu-berkeley dataset for robotic manipulation research," *The International Journal of Robotics Research*, vol. 36, no. 3, pp. 261–268, 2017. DOI: 10.1177/0278364917700714. eprint: https://doi.org/10.1177/0278364917700714. [Online]. Available: https://doi.org/10.1177/0278364917700714.

[7] B. Fang *et al.*, "Survey of imitation learning for robotic manipulation," *International Journal of Intelligent Robotics and Applications*, vol. 3, no. 4, pp. 362–369, Dec. 1, 2019, ISSN: 2366-598X. DOI: 10.1007/s41315-019-00103-5. [Online]. Available: https://doi.org/10.1007/s41315-019-00103-5.

[8] A. Radford *et al.*, *Learning transferable visual models from natural language supervision*, 2021. arXiv: 2103.00020 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2103.00020.

[9] A. Zeng *et al.*, *Transporter networks: Rearranging the visual world for robotic manipulation*, 2022. arXiv: 2010.14406 [cs.RO]. [Online]. Available: https://arxiv.org/abs/2010.14406.

[10] A. Brohan *et al.*, *Rt-1: Robotics transformer for real-world control at scale*, 2023. arXiv: 2212.06817 [cs.RO]. [Online]. Available: https://arxiv.org/abs/2212.06817.

[11] A. Brohan *et al.*, *Rt-2: Vision-language-action models transfer web knowledge to robotic control*, 2023. arXiv: 2307.15818 [cs.RO]. [Online]. Available: https://arxiv.org/abs/2307.15818.

[12] S. Liu *et al.*, *Rdt-1b: A diffusion foundation model for bimanual manipulation*, 2025. arXiv: 2410.07864 [cs.RO]. [Online]. Available: https://arxiv.org/abs/2410.07864.

[13] M. J. Kim *et al.*, *Openvla: An open-source vision-language-action model*, 2024. arXiv: 2406.09246 [cs.RO]. [Online]. Available: https://arxiv.org/abs/2406.09246.

[14] O. M. Team *et al.*, *Octo: An open-source generalist robot policy*, 2024. arXiv: 2405.12213 [cs.RO]. [Online]. Available: https://arxiv.org/abs/2405.12213.

[15] E. Collaboration *et al.*, *Open x-embodiment: Robotic learning datasets and rt-x models*, 2024. arXiv: 2310.08864 [cs.RO]. [Online]. Available: https://arxiv.org/abs/2310.08864.

[16] A. Rajeswaran *et al.*, *Learning complex dexterous manipulation with deep reinforcement learning and demonstrations*, 2018. arXiv: 1709.10087 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1709.10087.

[17] S. James and A. J. Davison, *Q-attention: Enabling efficient learning for vision-based robotic manipulation*, 2022. arXiv: 2105.14829 [cs.RO]. [Online]. Available: https://arxiv.org/abs/2105.14829.

[18] R. Jangir, G. Alenyà, and C. Torras, "Dynamic cloth manipulation with deep reinforcement learning," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 4630–4636. DOI: 10.1109/ICRA40945.2020.9196659.

[19] J. Matas, S. James, and A. J. Davison, "Sim-to-real reinforcement learning for deformable object manipulation," in *Proceedings of The 2nd Conference on Robot Learning*, A. Billard *et al.*, Eds., ser. Proceedings of Machine Learning Research, vol. 87, PMLR, 29–31 Oct 2018, pp. 734–743. [Online]. Available: https://proceedings.mlr.press/v87/matas18a.html.

[20] A. Kumar *et al.*, *Rma: Rapid motor adaptation for legged robots*, 2021. arXiv: 2107.04034 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2107.04034.

[21] J. Tobin *et al.*, *Domain randomization for transferring deep neural networks from simulation to the real world*, 2017. arXiv: 1703.06907 [cs.RO]. [Online]. Available: https://arxiv.org/abs/1703.06907.

[22] OpenAI *et al.*, *Solving rubik's cube with a robot hand*, 2019. arXiv: 1910.07113 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1910.07113.

[23] C. Smith *et al.*, "Dual arm manipulation—a survey," *Robotics and Autonomous Systems*, vol. 60, no. 10, pp. 1340–1353, 2012, ISSN: 0921-8890. DOI: https://doi.org/10.1016/j.robot.2012.07.005. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S092188901200108X.

[24] I.-C. A. Liu *et al.*, "Voxact-b: Voxel-based acting and stabilizing policy for bimanual manipulation," in *Conference on Robot Learning*, 2024.

[25] M. Grotz *et al.*, *Peract2: Benchmarking and learning for robotic bimanual manipulation tasks*, 2024. arXiv: 2407.00278 [cs.RO]. [Online]. Available: https://arxiv.org/abs/2407.00278.

[26] T. Z. Zhao *et al.*, *Learning fine-grained bimanual manipulation with low-cost hardware*, 2023. arXiv: 2304.13705 [cs.RO]. [Online]. Available: https://arxiv.org/abs/2304.13705.

[27] F. Xie *et al.*, *Deep imitation learning for bimanual robotic manipulation*, 2020. arXiv: 2010.05134 [cs.RO]. [Online]. Available: https://arxiv.org/abs/2010.05134.

[28] T. Lin *et al.*, *Sim-to-real reinforcement learning for vision-based dexterous manipulation on humanoids*, 2025. arXiv: 2502.20396 [cs.RO]. [Online]. Available: https://arxiv.org/abs/2502.20396.

[29] Y. Chen *et al.*, *Towards human-level bimanual dexterous manipulation with reinforcement learning*, 2022. arXiv: 2206.08686 [cs.RO]. [Online]. Available: https://arxiv.org/abs/2206.08686.

[30] J. Grannen *et al.*, *Stabilize to act: Learning to coordinate for bimanual manipulation*, 2023. arXiv: 2309.01087 [cs.RO]. [Online]. Available: https://arxiv.org/abs/2309.01087.

[31] T. Xie *et al.*, *Text2reward: Reward shaping with language models for reinforcement learning*, 2024. arXiv: 2309.11489 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2309.11489.

[32] W. Yu *et al.*, *Language to rewards for robotic skill synthesis*, 2023. arXiv: 2306.08647 [cs.RO]. [Online]. Available: https://arxiv.org/abs/2306.08647.

[33] Y. Wang *et al.*, *Robogen: Towards unleashing infinite data for automated robot learning via generative simulation*, 2024. arXiv: 2311.01455 [cs.RO]. [Online]. Available: https://arxiv.org/abs/2311.01455.

[34] J. Rocamonde *et al.*, *Vision-language models are zero-shot reward models for reinforcement learn-*

ing, 2024. arXiv: 2310.12921 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2310.12921.

[35] Y. Cui *et al.*, *Can foundation models perform zero-shot task specification for robot manipulation?* 2022. arXiv: 2204.11134 [cs.RO]. [Online]. Available: https://arxiv.org/abs/2204.11134.

[36] P. Christiano *et al.*, *Deep reinforcement learning from human preferences*, 2023. arXiv: 1706.03741 [stat.ML]. [Online]. Available: https://arxiv.org/abs/1706.03741.

[37] H. Lee *et al.*, *Rlaif vs. rlhf: Scaling reinforcement learning from human feedback with ai feedback*, 2024. arXiv: 2309.00267 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2309.00267.

[38] Y. Bai *et al.*, *Constitutional ai: Harmlessness from ai feedback*, 2022. arXiv: 2212.08073 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2212.08073.

[39] R. Sutton and A. Barto, "Reinforcement learning: An introduction," *IEEE Transactions on Neural Networks*, vol. 9, no. 5, pp. 1054–1054, 1998. DOI: 10.1109/TNN.1998.712192.

[40] J. Schulman *et al.*, *Proximal policy optimization algorithms*, 2017. arXiv: 1707.06347 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1707.06347.

[41] T. Haarnoja *et al.*, *Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor*, 2018. arXiv: 1801.01290 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1801.01290.

[42] Y. Liang, K. Ellis, and J. Henriques, "Rapid motor adaptation for robotic manipulator arms," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2024, pp. 16 404–16 413.

[43] A. Kumar *et al.*, *Adapting rapid motor adaptation for bipedal robots*, 2022. arXiv: 2205.15299 [cs.RO]. [Online]. Available: https://arxiv.org/abs/2205.15299.

[44] Y. Xiang *et al.*, *Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes*, 2018. arXiv: 1711.00199 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1711.00199.

[45] S. Tao *et al.*, *Maniskill3: Gpu parallelized robotics simulation and rendering for generalizable embodied ai*, 2024. arXiv: 2410.00425 [cs.RO]. [Online]. Available: https://arxiv.org/abs/2410.00425.

[46] F. Xiang *et al.*, *Sapien: A simulated part-based interactive environment*, 2020. arXiv: 2003.08515 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2003.08515.

[47] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, 2017. arXiv: 1412.6980 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1412.6980.

[48] A. Paszke *et al.*, *Pytorch: An imperative style, high-performance deep learning library*, 2019. arXiv: 1912.01703 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1912.01703.

[49] K. Lee, L. Smith, and P. Abbeel, *Pebble: Feedback-efficient interactive reinforcement learning via relabeling experience and unsupervised pre-training*, 2021. arXiv: 2106.05091 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2106.05091.

[50] R. A. Bradley and M. E. Terry, "Rank analysis of incomplete block designs: I. the method of paired comparisons," *Biometrika*, vol. 39, p. 324, 1952. [Online]. Available: https://api.semanticscholar.org/CorpusID:125209808.

[51] T. Kaufmann *et al.*, *A survey of reinforcement learning from human feedback*, 2024. arXiv: 2312.14925 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2312.14925.

[52] P.-Y. Oudeyer, F. Kaplan, and V. V. Hafner, "Intrinsic motivation systems for autonomous mental development," *IEEE Transactions on Evolutionary Computation*, vol. 11, no. 2, pp. 265–286, 2007. DOI: 10.1109/TEVC.2006.890271.

[53] J. Beirlant *et al.*, "Nonparametric entropy estimation: An overview," *International Journal of Mathematical and Statistical Sciences*, vol. 6, Jan. 1997.

[54] H. Singh *et al.*, "Nearest neighbor estimates of entropy," *American Journal of Mathematical and Management Sciences*, vol. 23, no. 3-4, pp. 301–321, 2003. DOI: 10.1080/01966324.2003.10737616. eprint: https://doi.org/10.1080/01966324.2003.10737616. [Online]. Available: https://doi.org/10.1080/01966324.2003.10737616.

[55] R. Akrour, M. Schoenauer, and M. Sebag, *April: Active preference-learning based reinforcement learning*, 2012. arXiv: 1208.0984 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1208.0984.

[56] J. Schulman *et al.*, *Trust region policy optimization*, 2017. arXiv: 1502.05477 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1502.05477.

[57] V. Mnih *et al.*, *Asynchronous methods for deep reinforcement learning*, 2016. arXiv: 1602.01783 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1602.01783.

[58] K. Lee *et al.*, *B-pref: Benchmarking preference-based reinforcement learning*, 2021. arXiv: 2111.03026 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2111.03026.

[59] Y. Wang *et al.*, *Rl-vlm-f: Reinforcement learning from vision language foundation model feedback*, 2024. arXiv: 2402.03681 [cs.RO]. [Online]. Available: https://arxiv.org/abs/2402.03681.

Department of Engineering Science

4YP Risk Assessment 2022

UNIVERSITY OF OXFORD

Description of 4YP task or aspect being risk assessed here: *(Read the Guidance Notes before completing this form)*		4YP Project Number:
Language-conditioned and generalisable robotic manipulation with reinforcement learning		13323
Site, Building & Room Number: Information Engineering Building, 2nd floor, V66 office	Other Relevant risk Assessments: computer-based risk assessment	
Assessment undertaken by: Seung-Bin Joo	Signed:	Date: 06/11/2024
Assessment Supervisor: João Henriques	Signed:	Date: 7/11/2024

Assessing the Risk*

You can do this for each hazard as follows:

- Consequences: Decide how severe the outcome for each hazard would be if something went wrong (i.e. what are the Consequences?) Death would be "Severe", a minor cut to a finger could be regarded as "Insignificant".
- Likelihood: How likely are these Consequences to actually happen? Highly likely? Remotely likely, or somewhere in between?
- Risk Rating: Start at the left of the coloured Matrix. On your chosen Consequences row, read across until you are in the correct Likelihood column for the hazard in question. For example, an outcome with Severe consequences but with a Low probability of actually happening equates to a Medium risk overall. In this case "Medium" is what should be written in the Risk.

RISK MATRIX

CONSEQUENCES	LIKELIHOOD (or probability)			
	High	**Medium**	**Low**	**Remote**
Severe	**High**	**High**	**Medium**	**Low**
Moderate	**High**	**Medium**	**Medium/Low**	**Effectively Zero**
Insignificant	**Medium/Low**	**Low**	**Low**	**Effectively Zero**
Negligible	**Effectively Zero**	**Effectively Zero**	**Effectively Zero**	**Effectively Zero**

Department of Engineering Science

Supplementary Questions for 4th Year Project Students

Risk Factor	Answer	Things to Consider	Record details here
Has the checklist covered all the problems that may arise from working with the VDU?	☑ Yes ☐ No		
Are you free from experiencing any fatigue, stress, discomfort or other symptoms which you attribute to working with the VDU or work environment?	☑ Yes ☐ No	Any aches, pains or sensory loss (tingling or pins and needles) in your neck, back shoulders or upper limbs. Do you experience restricted joint movement, impaired finger movements, grip or other disability, temporary or permanently	
Do you take adequate breaks when working at the VDU?	☑ Yes ☐ No	Periods of two minutes looking away from the screen taken every 20 minutes and longer periods every 2 hours Natural breaks for taking a drink and moving around the office answering the phone etc.	
How many hours per day do you spend working with this computer?	☐ 1-2 ☐ 3-4 ☑ 5-7 ☐ 8 or more		
How many days per week do you spend working with this computer?	☐ 1-2 ☐ 3-5 ☑ 6-7		
Please describe your computer usage pattern	A fair amount of work (including all of 4YP work) is done on computer. 4 ~ 7 hours of computer work on most days. Rests are taken every 1 ~ 2 hours.		